Бесплатная электронная книга

УЧУСЬ

meteor

Free unaffiliated eBook created from
Stack Overflow contributors.

#meteor

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: meteor

It is an unofficial and free meteor ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official meteor.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с метеоритом

замечания

Meteor - это **полнотекстовая** платформа JavaScript для разработки современных веб-приложений и мобильных приложений.

В рамках *одного* проекта вы можете создать своего клиента (браузер и / или гибридное мобильное приложение для Android и / или iOS) *и* сторон сервера.

Справочные страницы:

- Руководство Метеор
- Документы API Meteor
- Учебники Meteor
- Метеор форум

Версии

Версия	Дата выхода
0.4.0	2012-08-30
0.5.0	2013-10-17
0.6.0	2013-04-04
0.7.0	2013-12-20
0.8.0	2014-04-21
0.9.0	2014-08-26
0.9.1	2014-09-04
0.9.2	2014-09-15
0.9.3	2014-09-25
0.9.4	2014-10-13
1.0.1	2014-12-09
1.0.2	2014-12-19
1.0.3.1	2014-12-09

Версия	Дата выхода
1.1.0	2015-03-31
1.2.0	2015-09-21
1.3.0	2016-03-27
1.4.0	2016-07-25
1.5.0	2017-05-30

Examples

Начиная

Установить Метеор

В OS X и Linux

Установите последний официальный релиз Meteor с вашего терминала:

```
$ curl https://install.meteor.com/ | sh
```

В Windows

Загрузите официальный установщик Meteor здесь .

Создайте приложение

После установки Meteor создайте проект:

```
$ meteor create myapp
```

Запустить его

Запустить его локально:

```
$ cd myapp
$ meteor npm install
$ meteor
```

Примечание. Сервер Meteor работает на: http: // localhost: 3000 /

Затем отправляйтесь в http: // localhost: 3000, чтобы увидеть новое приложение Meteor.

- Узнайте больше о том, как начать работу с Метеор в « Метеорном гиде» .
- Исследуйте пакеты Meteor в атмосфере - современный, быстрый, хорошо спроектированный менеджер пакетов.

Примеры приложений

В Meteor встроено несколько примеров приложений. Вы можете создать проект с одним из них и узнать, как он был построен. Чтобы создать пример приложения, установите Meteor (см. Начало работы), а затем введите:

```
meteor create --example <app name>
```

Например, чтобы создать образец приложения `todos` , напишите:

```
meteor create --example todos
```

Чтобы получить список всех примеров приложений, введите:

```
meteor create --list
```

Управление пакетами

У Meteor есть собственный репозиторий пакетов на сайте weatherjs.com

Вы можете добавить новые пакеты из атмосферы, запустив:

```
meteor add [package-author-name:package-name]
```

Например:

```
meteor add kadira:flow-router
```

Аналогичным образом вы можете удалить один и тот же пакет:

```
meteor remove kadira:flow-router
```

Чтобы увидеть текущие пакеты в вашем проекте, введите:

```
meteor list
```

Список пакетов также можно найти в файле `./meteor/packages` . Чтобы добавить пакет, добавьте имя пакета в этот файл и удалите его.

Чтобы добавить пакет локально (например, неопубликованные пакеты или отредактированная версия опубликованных пакетов), сохраните пакет в папке `packages` в корне.

Начиная с версии 1.3, Meteor **добавила поддержку пакетов npm** .

Вы можете использовать команду `npm` в каталоге проекта Meteor, как обычно бываете без Meteor, или с помощью команды `meteor npm` , которая будет использовать пакетную версию npm.

Понимание прогресса сборки

Иногда сборка занимает больше времени, чем ожидалось. Есть несколько переменных среды, которые вы можете установить, чтобы лучше понять, что происходит во время процесса сборки.

```
METEOR_DEBUG_BUILD=1       (logs progress)
METEOR_PROFILE=<n>         (logs time spent)
METEOR_DEBUG_SPRINGBOARD=1 (?)
METEOR_DEBUG_SQL=1         (logs SQLITE calls)
METEOR_PROGRESS_DEBUG=1    (? looks like it might be useful, but seems confusing)
```

Где `<n>` - количество мс. Любой процесс, занимающий больше времени, будет зарегистрирован.

Пример Linux / OSX

```
export METEOR_DEBUG_BUILD=1
export METEOR_PROFILE=100
meteor
```

Пример Windows

```
set METEOR_DEBUG_BUILD=1
set METEOR_PROFILE=100
meteor
```

Проверка версии Meteor Tool & Meteor Projects

Метеорный инструмент

Чтобы проверить установленную версию инструмента Meteor, просто запустите следующую команду за пределами любых проектов Meteor:

```
meteor --version
```

Чтобы получить список всех официальных (рекомендуемых) релизов Meteor, запустите:

```
meteor show METEOR
```

Метеорные проекты

Если вы хотите проверить версию проекта Meteor, вы также можете выполнить следующую команду внутри проекта:

```
meteor --version
```

или просто распечатать содержимое файла `.meteor/release` :

```
cat .meteor/release
```

Если вы хотите проверить версию пакетов, которые в настоящее время установлены в вашем проекте Meteor, распечатайте содержимое файла. `.meteor/versions` :

```
cat .meteor/versions
```

Метеорологический сайт

Чтобы посмотреть, какая версия веб-сайта Meteor a Meteor работает, выгрузите содержимое `Meteor.release` в консоли браузера во время посещения веб-сайта:

```
Meteor.release
```

Обновление проектов и установленных пакетов Meteor

Meteor Tool уведомит вас, когда появится более новая версия.

Чтобы обновить проекты Meteor до последней версии, выполните следующую команду внутри проекта Meteor:

```
meteor update
```

Если вы хотите обновить свой проект Meteor до определенного выпуска Meteor, выполните следующую команду внутри проекта:

```
meteor update --release <release>
```

Если вы хотите обновить все непрофильные пакеты, запустите:

```
meteor update --packages-only
```

Вы также можете обновлять определенные пакеты, передавая их имена в качестве аргумента командной строки для `meteor update` , например:

```
meteor update [packageName packageName2 ...]
```

Создайте мобильные приложения

Meteor использует Cordova для упаковки вашего приложения в *гибридное* мобильное приложение. После упаковки приложение можно распространять как собственные приложения (через Apple App Store, Google Play Store и т. Д.),

1. **Добавьте** целевую платформу (ы) в проект Meteor:

```
meteor add-platform android
meteor add-platform ios # Only available with Mac OS
```

2. **Установите** Android SDK и / или Xcode (для iOS, требуется Mac OS).

3. **Запустите** проект (начните с режима разработки):

```
meteor run android # You may need to configure a default Android emulator first
```

Для iOS (доступно только для Mac OS):

```
meteor run ios # This will auto start an iOS simulator
```

4. **Создайте** свой пакет приложений для распространения:

```
meteor build <output_folder> --server <url_app_should_connect_to>
```

Это создаст папки `android` и / или `ios` вместе с вашим комплектом серверов.

- В папке `android` содержится файл `release-unsigned.apk` который вам нужно подписать и закрепить почтой.

- Папка `ios` содержит проект Xcode, который нужно подписать.

См. Также раздел « Приложения для мобильных приложений» .
Справочная страница: Meteor Guide> Build> Mobile

Прочитайте Начало работы с метеоритом онлайн: https://riptutorial.com/ru/meteor/topic/439/начало-работы-с-метеоритом

глава 2: Blaze Templating

Вступление

Blaze - мощная библиотека для создания пользовательских интерфейсов, написав динамические, реактивные HTML-шаблоны. Blaze templating позволяет использовать циклы и условную логику непосредственно в разметке HTML. В этом разделе объясняется и демонстрируется правильное использование шаблонов в Meteor.js с Blaze.

Examples

Заполнение шаблона при вызове метода

```
<template name="myTemplate">
  {{#each results}}
    <div><span>{{name}}</span><span>{{age}}</span></div>
  {{/each}}
</template>
```

```
Template.myTemplate.onCreated(function() {
  this.results = new ReactiveVar();
  Meteor.call('myMethod', (error, result) => {
    if (error) {
      // do something with the error
    } else {
      // results is an array of {name, age} objects
      this.results.set(result);
    }
  });
});

Template.myTemplate.helpers({
  results() {
    return Template.instance().results.get();
  }
});
```

Контекст данных шаблона

Всякий раз, когда вызывается шаблон, контекст данных по умолчанию шаблона неявно получен от вызывающего, как в примере childTemplate получает контекст данных parentTemplate, т. Е. Шаблон вызывающего абонента

```
<template name="parentTemplate">
    {{#with someHelperGettingDataForParentTemplate}}
    <h1>My name is {{firstname}} {{lastname}}</h1>
    //some stuffs here
    {{> childTemplate}}
    {{/with}}
```

```
</template>
```

В приведенной выше ситуации любые данные, получаемые вспомогательным экстрактом для родительского шаблона, автоматически получаются с помощью childTemplate. Например, к {{firstname}} и {{lastname}} можно получить доступ с childTemplate, как показано ниже.

```
<template name="childTemplate">
<h2>My name is also {{firstname}} {{lastname}}</h2>
</template>
```

Мы даже можем явно определить контекст данных childTemplate, передав аргументы шаблону, как показано ниже.

```
<template name="parentTemplate">
    {{#with someHelperGettingDataForParentTemplate}}
    <h1>My name is {{firstname}} {{lastname}}</h1>
    //some stuffs here
    {{> childTemplate childData=someHeplerReturningDataForChild}}
    {{/with}}
</template>
```

Предполагая, что helper **someHelperReturningDataForChild** возвращает объект, например {профессия: «Разработчик **метеоров** », хобби: «stackoverflowing»}, этот конкретный объект будет явным контекстом данных для childTemplate. Теперь в шаблоне для детей мы можем сделать что-то вроде

```
<template name="childTemplate">
    <h2>My profession is {{profession}}</h2>
    <h3>My hobby is {{hobby}}</h3>
</template>
```

Шаблоны Помощники

Шаблоны-помощники являются неотъемлемой частью Blaze и обеспечивают как бизнес-логику, так и реактивность на Шаблон. Важно помнить, что помощники шаблонов - это фактически реактивные вычисления , которые повторяются при каждом изменении их зависимостей. В зависимости от ваших потребностей помощники шаблонов могут быть определены глобально или ограничены определенным шаблоном. Ниже приведены примеры каждого подхода к определению вспомогательных элементов шаблона.

1. Пример помощника шаблона, привязанного к одному шаблону.

Сначала определите свой шаблон:

```
<template name="welcomeMessage">
  <h1>Welcome back {{fullName}}</h1>
</template>
```

Затем определите помощник шаблона. Это предполагает, что контекст данных шаблона содержит свойство firstName и lastName.

```
Template.welcomeMessage.helpers({
  fullName: function() {
    const instance = Template.instance();
    return instance.data.firstName + ' ' + instance.data.lastName
  },
});
```

2. Пример глобального помощника шаблона (этот помощник можно использовать из любого шаблона)

Сначала зарегистрируйте помощника:

```
Template.registerHelper('equals', function(item1, item2) {
  if (!item1 || !item2) {
    return false;
  }

  return item1 === item2;
});
```

С помощью помощника `equals` я могу теперь использовать его в любом шаблоне:

```
<template name="registration">
  {{#if equals currentUser.registrationStatus 'Pending'}}
    <p>Don't forget to complete your registration!<p>
  {{/if}}
</template>
```

Прочитайте Blaze Templating онлайн: https://riptutorial.com/ru/meteor/topic/2434/blaze-templating

глава 3: Electrify - компиляция Meteor как локально устанавливаемого приложения

Examples

Установка Electrify для приложения Meteor

Электронные порты HTML-приложений для веб-приложений для собственных приложений для ряда устройств, включая создание собственных настольных приложений. Это также очень легко начать!

Для начала у нас должны быть установлены `electron` , `nodejs` , `npm` , `git` и `meteor` . Знакомство с этими инструментами важно для работы с Meteor, поэтому убедитесь, что вы знаете об этих вещах в первую очередь.

электрон

```
npm install -g electrify
```

- `electron` - это то, что мы используем! Подробнее читайте здесь .
- `electrify` - это инструмент для упаковки приложений Meteor. Режим чтения здесь .

Другие требования по установке и использованию Electrify with Meteor

метеор

```
curl https://install.meteor.com/ | sh
```

Есть много способов установить Meteor, см. Здесь .

- `meteor` - это рамка JavaScript, которую мы будем использовать для создания нашего приложения. Это дает нам множество упрощений для некоторых довольно концептуально трудных проблем в веб-приложениях; его простота была отмечена как полезная для прототипических проектов. Подробнее читайте здесь .

NodeJS

```
apt-get install nodejs build-essentials
```

В зависимости от вашей ОС существует множество способов установки. Узнайте, в каком направлении вам нужно здесь .

- `nodejs` - это пакет для Node.js, который представляет собой среду Javascript для запуска JavaScript на стороне сервера. Подробнее читайте здесь .

НПМ

`npm` должен быть связан с установкой `nodejs` . Проверьте это, выполнив команду `npm -v` после установки `nodejs` .

- `npm` - диспетчер пакетов узлов. Это огромная коллекция модулей с открытым исходным кодом, которые вы можете легко добавить в свои проекты Node. Подробнее читайте здесь .

Использование электрификации при применении метеорита

Давайте загрузим пример проекта Meteor Todos с использованием сценария командной строки Linux (командной строки), чтобы проверить Электричество проекта в первый раз:

Требования к этому разделу:

Гит

```
apt-get install git-all
```

Существует множество способов установки Git. Проверьте их здесь .

- `git` - это система управления версиями файлов. Они могут храниться удаленно (т. Е. В Интернете) в публичных хранилищах (GitHub - довольно известный) или в частных репозиториях (например, BitBucket предоставляет ограниченные бесплатные частные хранилища). Подробнее [здесь] [5].

```
#!/usr/bin/bash

# Change this parameter to choose where to clone the repository to.
TODOSPATH="/home/user/development/meteor-todos"

# Download the repository to the $TODOSPATH location.
git clone https://github.com/meteor/todos.git "$TODOSPATH"

# Change directory (`cd`) into the Todos project folder.
cd "$TODOSPATH"
```

Теперь у нас должна быть папка проекта с именем «meteor-todos» в месте, указанном в параметре TODOSPATH. Мы также сменили каталог (`cd`) в папку проекта, поэтому давайте добавим Electrify в этот проект!

```
# It's really this simple.
electrify
```

Правильно - команда с одним словом, и наш проект готов. Разрешения могут вызывать ошибки при попытке запуска `electrify` как команды, в случае wihch попробуйте `sudo electrify` переопределить разрешения.

Тем не менее, попытайтесь разрешить эти проблемы с разрешениями - это не очень хорошая практика для излишней `sudo` (о которой я расскажу, но я мог бы написать целую другую тему о том, почему это так!)

Прочитайте Electrify - компиляция Meteor как локально устанавливаемого приложения онлайн: https://riptutorial.com/ru/meteor/topic/2526/electrify---компиляция-meteor-как-локально-устанавливаемого-приложения

глава 4: ESLint

Examples

Добавление eslint в проект Meteor

Мы будем использовать популярный `eslint-config-airbnb` как стартер, а также специальные правила Meteor, используя `eslint-import-resolver-meteor` .

Нам также необходимо установить `babel-parser` чтобы использовать функции ES7 с поддержкой Meteor, такие как async / await.

```
cd my-project
npm install --save-dev eslint-config-airbnb eslint-plugin-import eslint-plugin-react eslint-
plugin-jsx-a11y eslint babel-eslint eslint-import-resolver-meteor
touch .eslintrc.json
```

Затем просто используйте этот шаблон `.eslintrc.json` чтобы начать, вы можете переопределить правила по `.eslintrc.json` усмотрению.

```
{
  "parser": "babel-eslint",
  "settings": {
    "import/resolver": "meteor"
  },
  "extends": "airbnb",
  "rules": {}
}
```

Использование скрипта npm для ввода кода

Измените свой `package.json` чтобы добавить следующий скрипт:

```
{
  "scripts": {
    "lint": "eslint .;exit 0"
  }
}
```

Затем запустите его, используя `npm run lint`

Мы используем `exit 0` в качестве трюка, чтобы изящно завершить сценарий, когда linting терпит неудачу, иначе `npm` будет использовать `eslint` возврата `eslint` и сбой.

Прочитайте ESLint онлайн: https://riptutorial.com/ru/meteor/topic/3772/eslint

глава 5: Meteor + React + ReactRouter

Вступление

В этом документе будет показано, как использовать ReactRouter с Meteor и React. От нуля до рабочего приложения, включая роли и аутентификацию.

Я покажу каждый шаг с примером

1- Создать проект

2- Add React + ReactRouter

3- Добавить учетные записи

4- Добавить пакеты ролей

Examples

Создать проект

1- Сначала установите https://www.meteor.com/install

2. Создайте проект. (`--bare` - создать пустой проект)

```
meteor create --bare MyAwesomeProject
```

3- Создайте минимальную структуру файла (`-p` для создания промежуточных каталогов):

```
cd MyAwesomeProject

mkdir -p client server imports/api imports/ui/{components,layouts,pages}
imports/startup/{client,server}
```

4 Теперь создайте файл HTML в файле client / main.html

```
<head>
   <meta charset="utf-8">
   <title>My Awesome Meteor_React_ReactRouter_Roles App</title>
</head>

<body>
  Welcome to my Meteor_React_ReactRouter_Roles app
</body>
```

5- Убедитесь, что он работает: (3000 - это порт по умолчанию, поэтому вы можете пропустить «-p 3000»)

```
meteor run -p 3000
```

и открытие вашего браузера на «localhost: 3000»

Замечания:

- Я пропускаю некоторые другие файлы, которые вам нужно создать, чтобы сделать вещи короче. В частности, вам нужно будет создать некоторые файлы index.js в *клиенте* , import */ startup / {client, server}* и *серверных* каталогах.
- Вы можете просмотреть полный пример в https://github.com/rafa-lft/Meteor_React_Base . Найдите тег *Step1_CreateProject*

Добавить React + ReactRouter

При необходимости перейдите в каталог проекта `cd MyAwesomeProject`

1- Добавить реактивный и реактивный маршрутизатор

```
meteor npm install --save react-router@3.0.0 react@15.5.4 react-dom@15.5.4
```

2- Измените client / main.html и замените контент:

```
<body>
   <div id="react-root"></div>
</body>
```

Независимо от того, что решает реактор, он покажет его в элементе «# response-root»

3- Создайте файл макетов в импорте / ui / layouts / App.jsx

```
import React, { Component } from 'react';
import PropTypes from 'prop-types';


class App extends Component {
  constructor(props) {
    super(props);
  }

  render() {
    return (
      <div>
        {this.props.children}
      </div>
    );
  }
}

App.propTypes = {
  children: PropTypes.node
};
```

```
export default App;
```

4- Создать файл маршрутов в импорте / запуске / клиенте / Routes.jsx

```
import ReactDOM from 'react-dom';
import React, { Component } from 'react';
import { Router, Route, IndexRoute, browserHistory } from 'react-router';

import App from '../../ui/layouts/App.jsx';

import NotFound from '../../ui/pages/NotFound.jsx';
import Index from '../../ui/pages/Index.jsx';


class Routes extends Component {
  constructor(props) {
    super(props);
  }

  render() {
    return (
      <Router history={ browserHistory }>
        <Route path="/" component={ App }>
          <IndexRoute name="index" component={ Index }/>
          <Route path="*" component={ NotFound }/>
        </Route>
      </Router>
    );
  }
}

Routes.propTypes = {};


Meteor.startup(() =>{
  ReactDOM.render(
    <Routes/>,
    document.getElementById('react-root')
  );
});
```

Замечания:

- Я пропускаю некоторые другие файлы, которые вам нужно создать, чтобы сделать вещи короче. В частности, проверьте импорт / ui / pages {Index.jsx, NotFound.jsx}.
- Вы можете просмотреть полный пример в https://github.com/rafa-lft/Meteor_React_Base . Найдите тег *Step2_ReactRouter*

Шаг 3. Добавление учетных записей

При необходимости перейдите в каталог проекта `cd MyAwesomeProject`

1- Добавление пакетов учетных записей: `meteor add accounts-base accounts-password react-meteor-data`

2- Добавьте маршруты для *входа* и *регистрации* в импорте / запуске / Routes.jsx. Метод *render ()* будет выглядеть следующим образом:

```
  render() {
    return (
      <Router history={ browserHistory }>
        <Route path="/" component={ App }>
          <IndexRoute name="index" component={ Index }/>
          <Route name="login" path="/login" component={ Login }/>
          <Route name="signup" path="/signup" component={ Signup }/>
          <Route name="users" path="/users" component={ Users }/>
          <Route name="editUser" path="/users/:userId" component={ EditUser }/>
          <Route path="*" component={ NotFound }/>
        </Route>
      </Router>
    );
  }
```

Замечания:

- Я пропущу некоторые другие файлы, которые вам понадобятся, чтобы сделать вещи короче. В частности, проверьте импорт / startup / server / index.js import / ui / layouts / {App, NavBar} .jsx и import / ui / pages / {Вход, регистрация, пользователи, EditUser} .jsx
- Вы можете просмотреть полный пример в https://github.com/rafa-lft/Meteor_React_Base . Искать тег *Step3_Accounts*

Добавить роли

1- Добавить пакет ролей (https://github.com/alanning/meteor-roles)

```
meteor add alanning:roles
```

2- Создайте несколько констант ролей. В файлах импорта / api / accounts / role.js

```
const ROLES = {
  ROLE1: 'ROLE1',
  ROLE2: 'ROLE2',
  ADMIN: 'ADMIN'
};

export default ROLES;
```

3- Я не буду показывать, как добавлять / обновлять роли для пользователя, просто упомянем, что на стороне сервера вы можете установить роли пользователя с помощью `Roles.setUserRoles(user.id, roles);` Проверьте дополнительную информацию в https://github.com/alanning/meteor-roles и http://alanning.github.io/meteor-

roles/classes/Roles.html.

4- Предполагая, что вы уже настроили все файлы учетных записей и ролей (см. Полный пример в https://github.com/rafa-lft/Meteor_React_Base . Ищите тег *Step4_roles*), теперь мы можем создать метод, который будет отвечать за разрешение (или нет) доступ к различным маршрутам. В импорте / запуске / клиенте / Routes.jsx

```
class Routes extends Component {
  constructor(props) {
    super(props);
  }

  authenticate(roles, nextState, replace) {
    if (!Meteor.loggingIn() && !Meteor.userId()) {
      replace({
        pathname: '/login',
        state: {nextPathname: nextState.location.pathname}
      });
      return;
    }
    if ('*' === roles) { // allow any logged user
      return;
    }
    let rolesArr = roles;
    if (!_.isArray(roles)) {
      rolesArr = [roles];
    }
    // rolesArr = _.union(rolesArr, [ROLES.ADMIN]);// so ADMIN has access to everything
    if (!Roles.userIsInRole(Meteor.userId(), rolesArr)) {
      replace({
        pathname: '/forbidden',
        state: {nextPathname: nextState.location.pathname}
      });
    }
  }

  render() {
    return (
      <Router history={ browserHistory }>
        <Route path="/" component={ App }>
          <IndexRoute name="index" component={ Index }/>
          <Route name="login" path="/login" component={ Login }/>
          <Route name="signup" path="/signup" component={ Signup }/>

          <Route name="users" path="/users" component={ Users }/>

          <Route name="editUser" path="/users/:userId" component={ EditUser }
                 onEnter={_.partial(this.authenticate, ROLES.ADMIN)} />


          {/* ********************
           Below links are there to show Roles authentication usage.
           Note that you can NOT hide them by
           { Meteor.user() && Roles.userIsInRole(Meteor.user(), ROLES.ROLE1) &&
           <Route name=.....
           }
           as doing so will change the Router component on render(), and ReactRouter will
complain with:
```

```
           Warning: [react-router] You cannot change <Router routes>; it will be ignored

           Instead, you can/should hide them on the NavBar.jsx component... don't worry: if
someone tries to access
           them, they will receive the Forbidden.jsx component
           ************/ }
          <Route name="forAnyOne" path="/for_any_one" component={ ForAnyone }/>

          <Route name="forLoggedOnes" path="/for_logged_ones" component={ ForLoggedOnes }
                 onEnter={_.partial(this.authenticate, '*')} />

          <Route name="forAnyRole" path="/for_any_role" component={ ForAnyRole }
                 onEnter={_.partial(this.authenticate, _.keys(ROLES))}/>

          <Route name="forRole1or2" path="/for_role_1_or_2" component={ ForRole1or2 }
                 onEnter={_.partial(this.authenticate, [ROLES.ROLE1, ROLES.ROLE2])} />

          <Route name="forRole1" path="/for_role1" component={ ForRole1 }
                 onEnter={_.partial(this.authenticate, ROLES.ROLE1)}/>

          <Route name="forRole2" path="/for_role2" component={ ForRole2 }
                 onEnter={_.partial(this.authenticate, ROLES.ROLE2)} />


          <Route name="forbidden" path="/forbidden" component={ Forbidden }/>

          <Route path="*" component={ NotFound }/>
        </Route>
      </Router>
    );
  }
}
```

Мы добавили триггер *onEnter* для некоторых маршрутов. Для этих маршрутов мы также передаем, какие роли разрешены для входа. Обратите внимание, что обратный вызов onEnter изначально получает 2 параметра. Мы используем частичное выражение underscore (http://underscorejs.org/#partial) , чтобы добавить еще один (роли). Метод *аутентификации* (вызываемый onEnter) получает роли и:

- Убедитесь, что пользователь вошел в систему. Если нет, перенаправляется на '/ login'.
- Если роли === '*', мы предполагаем, что любой зарегистрированный пользователь может войти, поэтому мы разрешаем это
- Кроме того, мы проверяем, разрешен ли пользователь (Roles.userIsInRole), а если нет, мы перенаправляем на запрещенный.
- При желании вы можете раскомментировать строку, поэтому ADMIN имеет доступ ко всему.

В коде есть несколько примеров различных маршрутов, которые разрешены для кого-либо (без обратного вызова onEnter), для любого зарегистрированного пользователя, для любого зарегистрированного пользователя с хотя бы одной ролью и для определенных ролей.

Также обратите внимание, что ReactRouter (по крайней мере, на версии 3) не позволяет модифицировать маршруты на Render. Таким образом, вы не можете скрыть маршруты в Routes.jsx. По этой причине мы перенаправляем / запрещаем метод аутентификации.

5- Общая ошибка с ReactRouter и Meteor, относится к обновлениям статуса пользователя, которые не отображаются. Например, пользователь вышел из системы, но мы все еще показываем его имя на навигационной панели. Это происходит потому, что Meteor.user () изменился, но мы не перерисовываем.

Эта ошибка может быть решена путем вызова Meteor.user () в createContainer. Вот пример этого, используемый в импорте / ui / layouts / NavBar.jsx:

```
export default createContainer((/* {params}*/) =>{
  Meteor.user(); // so we render again in logout or if any change on our User (ie: new roles)
  const loading = !subscription.ready();
  return {subscriptions: [subscription], loading};
}, NavBar);
```

Замечания:

- Я пропущу некоторые другие файлы, которые вам понадобятся, чтобы сделать вещи короче. В частности, проверьте импорт / startup / server / index.js import / ui / layouts / {App, NavBar} .jsx и import / ui / pages / {Вход, регистрация, пользователи, EditUser} .jsx
- Вы можете просмотреть полный пример в https://github.com/rafa-lft/Meteor_React_Base . Искать метки *Step4_roles*

Прочитайте Meteor + React + ReactRouter онлайн:
https://riptutorial.com/ru/meteor/topic/10114/meteor-plus-react-plus-reactrouter

глава 6: MongoDB

Вступление

MongoDB - бесплатная и открытая кросс-платформенная программа для документирования документов. В отличие от классических баз данных SQL, MongoDB использует BSON (например, JSON) для хранения данных. Meteor был разработан для использования MongoDB для хранения базы данных, и в этом разделе объясняется, как реализовать хранилище MongoDB в приложениях Meteor.

Examples

Экспорт удаленной базы данных Mongo DB, импорт в локальную базу данных Meteor Mongo DB

Полезно, когда вы хотите захватить копию производственной базы данных, чтобы поиграть с локально.

1. `mongodump --host some-mongo-host.com:1234 -d DATABASE_NAME -u DATABASE_USER -p DATABASE_PASSWORD` Это создаст локальную директорию `dump` ; внутри этого каталога вы увидите каталог с вашим `DATABASE_NAME` .
2. С помощью локального приложения meteor, изнутри каталога `dump` , запустите: `mongorestore --db meteor --drop -h localhost --port 3001 DATABASE_NAME`

Получите URL-адрес Mongo вашего локального метеора Mongo DB

Пока ваше приложение Meteor работает локально:

```
meteor mongo --url
```

Подключите локальное приложение Meteor к альтернативной базе данных Mongo DB

Установите `MONGO_URL` среды `MONGO_URL` перед запуском локального приложения Meteor.

Пример Linux / MacOS:

```
MONGO_URL="mongodb://some-mongo-host.com:1234/mydatabase" meteor
```

или же

```
export MONGO_URL="mongodb://some-mongo-host.com:1234/mydatabase"
```

```
meteor
```

Пример Windows

Примечание: не используйте "

```
set MONGO_URL=mongodb://some-mongo-host.com:1234/mydatabase
meteor
```

NPM

```
//package.json

"scripts": {
    "start": "MONGO_URL=mongodb://some-mongo-host.com:1234/mydatabase meteor"
}

$ npm start
```

Запуск Метеор без MongoDB

Установите `MONGO_URL` на любое произвольное значение, за исключением URL-адреса базы данных, и убедитесь, что в вашем проекте Meteor (включая коллекции, определенные пакетами Meteor) не определены коллекции, для запуска Meteor без MongoDB.

Обратите внимание, что без MongoDB методы сервер / клиент наряду с любыми пакетами, связанными с системой учетной записи пользователя Meteor, не будут определены. Пример: `Meteor.userId()`

Linux / Mac:

```
MONGO_URL="none" meteor
```

или же

```
export MONGO_URL="none"
meteor
```

Окна:

```
set MONGO_URL=none
meteor
```

Начиная

Вы можете запустить оболочку **`mongo`** , выполнив следующую команду внутри вашего

проекта Meteor:

```
meteor mongo
```

Обратите внимание: запуск серверной консоли базы данных работает только тогда, когда Meteor запускает приложение локально.

После этого вы можете перечислить все коллекции, выполнив следующую команду через оболочку `mongo` :

```
show collections
```

Вы также можете запускать базовые операции MongoDB, такие как запрос, вставка, обновление и удаление документов.

Документы запроса

Документы можно запросить с помощью метода `find()` , например:

```
db.collection.find({name: 'Matthias Eckhart'});
```

В нем будут перечислены все документы, для которых атрибут `name` установлен в `Matthias Eckhart` .

Вставка документов

Если вы хотите вставить документы в коллекцию, запустите:

```
db.collection.insert({name: 'Matthias Eckhart'});
```

Обновление документов

Если вы хотите обновлять документы, используйте метод `update()` , например:

```
db.collection.update({name: 'Matthias Eckhart'}, {$set: {name: 'John Doe'}});
```

Выполнение этой команды будет обновлять **один** документ, установив значение `John Doe` для поля `name` (изначально значение было `Matthias Eckhart`).

Если вы хотите обновить **все** документы, соответствующие определенным критериям, установите для параметра `multi` значение `true` , например:

```
db.collection.update({name: 'Matthias Eckhart'}, {$set: {name: 'John Doe'}}, {multi: true});
```

Теперь все документы в коллекции, которые первоначально атрибут `name` установленный в `Matthias Eckhart` , были обновлены до `John Doe` .

Удаление документов

Документы можно легко удалить с помощью метода `remove()` , например:

```
db.collection.remove({name: 'Matthias Eckhart'});
```

Это приведет к удалению всех документов, соответствующих значению, указанному в поле `name` .

Прочитайте MongoDB онлайн: https://riptutorial.com/ru/meteor/topic/1874/mongodb

глава 7: Nightwatch - Конфигурация и настройка

замечания

Nightwatch предоставляет приемные и сквозные тесты для приложений Meteor с v0.5 дней и управляет миграциями от PHP до Spark to Blaze и React; и все основные платформы непрерывной интеграции. Дополнительную помощь можно найти в:

Документация API Nightwatch
Группа пользователей Nightwatch.js Google

Examples

конфигурация

Основная причина, по которой Nightwatch настолько мощная, из-за этого отличный файл конфигурации. В отличие от большинства других платформ тестирования, Nightwatch полностью настраивается и настраивается в разных средах и технологиях.

.meteor / nightwatch.json

Следующий файл конфигурация для Метеора v1.3 и позже, а также поддерживает две среду ... а по `default` среду , которая запускает chromedriver браузер, и `phantom` среду , которая проходит испытания в обезглавленной среде.

```
{
  "nightwatch": {
    "version": "0.9.8"
  },
  "src_folders": [
    "./tests/nightwatch/walkthroughs"
  ],
  "custom_commands_path": [
    "./tests/nightwatch/commands"
  ],
  "custom_assertions_path": [
    "./tests/nightwatch/assertions"
  ],
  "output_folder": "./tests/nightwatch/reports",
  "page_objects_path": "./tests/nightwatch/pages",
  "globals_path": "./tests/nightwatch/globals.json",
  "selenium": {
    "start_process": true,
    "server_path": "./node_modules/starrynight/node_modules/selenium-server-standalone-
jar/jar/selenium-server-standalone-2.45.0.jar",
    "log_path": "tests/nightwatch/logs",
```

```
    "host": "127.0.0.1",
    "port": 4444,
    "cli_args": {
      "webdriver.chrome.driver":
"./node_modules/starrynight/node_modules/chromedriver/bin/chromedriver"
    }
  },
  "test_settings": {
    "default": {
      "launch_url": "http://localhost:5000",
      "selenium_host": "127.0.0.1",
      "selenium_port": 4444,
      "pathname": "/wd/hub",
      "silent": true,
      "disable_colors": false,
      "firefox_profile": false,
      "ie_driver": "",
      "screenshots": {
        "enabled": false,
        "path": "./tests/nightwatch/screenshots"
      },
      "desiredCapabilities": {
        "browserName": "chrome",
        "javascriptEnabled": true,
        "acceptSslCerts": true,
        "loggingPrefs": {
          "browser": "ALL"
        }
      },
      "exclude": "./tests/nightwatch/unittests/*",
      "persist_globals": true,
      "detailed_output": false
    },
    "phantom": {
      "desiredCapabilities": {
        "browserName": "phantomjs",
        "javascriptEnabled": true,
        "databaseEnabled": false,
        "locationContextEnabled": false,
        "applicationCacheEnabled": false,
        "browserConnectionEnabled": false,
        "webStorageEnabled": false,
        "acceptSslCerts": true,
        "rotatable": false,
        "nativeEvents": false,
        "phantomjs.binary.path": "./node_modules/starrynight/node_modules/phantomjs-
prebuilt/bin/phantomjs"
      }
    },
    "unittests": {
      "selenium": {
        "start_process": false,
        "start_session": false
      },
      "filter": "./tests/nightwatch/unittests/*",
      "exclude": ""
    }
  }
}
```

Установка и использование

Чтобы получить работу Nightwatch, вам понадобится локальная копия **селена,** которая является сервером управления и управления, который управляет автоматическими экземплярами браузера. Вам также понадобится веб-браузер, который может управлять селеном, например, **хромированный** или **фантомный** .

Добавьте в свой пакет `package.json` следующие devDependencies:

```
{
  "devDependencies": {
    "nightwatch": "0.9.8",
    "selenium-server-standalone-jar": "2.45.0",
    "chromedriver": "2.19.0",
    "phantomjs-prebuilt": "2.1.12"
  }
}
```

Затем установите все параметры.

```
cd myapp
meteor npm install
```

Затем вы можете запустить Nightwatch со следующими командами:

```
nightwatch -c .meteor/nightwatch.json
nightwatch -c .meteor/nightwatch.json --env phantom
```

Если вы еще не написали никаких тестов или не настроили структуру своей папки, вы можете получить некоторые ошибки.

Настройка сценариев запуска

В корне вашего приложения должен быть файл `package.json` , где вы можете определить скрипты и devDependencies.

```
{
  "name": "myapp",
  "version": "1.0.0",
  "scripts": {
    "start": "meteor --settings settings-development.json",
    "nightwatch": "nightwatch -c .meteor/nightwatch.json",
    "phantom": "nightwatch -c .meteor/nightwatch.json --env phantom",
  }
}
```

Затем вы сможете запустить ночной со следующим командам:

```
meteor npm run-script nightwatch
meteor npm run-script phantom
```

В этом примере было бы проще просто запустить `nightwatch -c .meteor/nightwatch.json` . Однако, с более сложными командами, со сложными переменными среды, параметрами и настройками, это становится очень полезным способом для определения сценариев devops для команды.

Структура папок

В базовой установке Nightwatch для Meteor будут установлены следующие каталоги и файлы.

```
/myapp
/myapp/.meteor/nightwatch.json
/client/main.html
/client/main.js
/client/main.css
/tests
/tests/nightwatch
/tests/nightwatch/assertions
/tests/nightwatch/commands
/tests/nightwatch/data
/tests/nightwatch/logs
/tests/nightwatch/pages
/tests/nightwatch/reports
/tests/nightwatch/screenshots
/tests/nightwatch/walkthroughs
/tests/nightwatch/walkthroughs/critical_path.js
/tests/nightwatch/globals.json
```

Тестирование данных

Nightwatch принимает второй `globals.json` конфигурации `globals.json` который вводит данные в сам тестовый бегун, очень похожий на то, как `Meteor.settings` делает данные из командной строки доступными во всем приложении.

globals.json

```
{
  "default" : {
    "url" : "http://localhost:3000",
    "user": {
      "name": "Jane Doe",
      "username" : "janedoe",
      "password" : "janedoe123",
      "email" : "janedoe@test.org",
      "userId": null
    }
  },
  "circle" : {
    "url" : "http://localhost:3000",
    "user": {
      "name": "Jane Doe",
      "username" : "janedoe",
      "password" : "janedoe123",
      "email" : "janedoe@test.org"
```

```
      "userId": null
    }
  },
  "galaxy" : {
    "url" : "http://myapp.meteorapp.com",
    "user": {
      "name": "Jane Doe",
      "username" : "janedoe",
      "password" : "janedoe123",
      "email" : "janedoe@test.org"
      "userId": null
    }
  }
}
```

Затем вы можете написать свои тесты, которые не жестко закодированы конкретными пользователями, паролями, входами поиска и т. Д.

```
module.exports = {
  "Login App" : function (client) {
    client
      .url(client.globals.url)
      .login(client.globals.user.email, client.globals.user.password)
      .end();
  }
};
```

Прочитайте Nightwatch - Конфигурация и настройка онлайн:
https://riptutorial.com/ru/meteor/topic/5901/nightwatch---конфигурация-и-настройка

глава 8: Агрегация MongoDB

замечания

Агрегация серверов

Средние агрегирующие запросы в Метеор

возможно ли упаковать «настоящую» библиотеку mongodb для использования на стороне * сервера * только в метеор 0.6

Агрегирование клиентов (Minimongo)

https://github.com/utunga/pocketmeteor/tree/master/packages/mongowrapper

Examples

Агрегация серверов

Решение Эндрю Мао. Средние агрегирующие запросы в Метеор

```
Meteor.publish("someAggregation", function (args) {
    var sub = this;
    // This works for Meteor 0.6.5
    var db = MongoInternals.defaultRemoteCollectionDriver().mongo.db;

    // Your arguments to Mongo's aggregation. Make these however you want.
    var pipeline = [
        { $match: doSomethingWith(args) },
        { $group: {
            _id: whatWeAreGroupingWith(args),
            count: { $sum: 1 }
        }}
    ];

    db.collection("server_collection_name").aggregate(
        pipeline,
        // Need to wrap the callback so it gets called in a Fiber.
        Meteor.bindEnvironment(
            function(err, result) {
                // Add each of the results to the subscription.
                _.each(result, function(e) {
                    // Generate a random disposable id for aggregated documents
                    sub.added("client_collection_name", Random.id(), {
                        key: e._id.somethingOfInterest,
                        count: e.count
                    });
                });
                sub.ready();
            },
            function(error) {
                Meteor._debug( "Error doing aggregation: " + error);
```

```
            }
        )
    );
});
```

Агрегация в методе сервера

Другой способ выполнения агрегаций - использовать `Mongo.Collection#rawCollection()`

Это можно запустить только на сервере.

Вот пример, который вы можете использовать в Meteor 1.3 и выше:

```
Meteor.methods({
   'aggregateUsers'(someId) {
      const collection = MyCollection.rawCollection()
      const aggregate = Meteor.wrapAsync(collection.aggregate, collection)

      const match = { age: { $gte: 25 } }
      const group = { _id:'$age', totalUsers: { $sum: 1 } }

      const results = aggregate([
         { $match: match },
         { $group: group }
      ])

      return results
   }
})
```

Прочитайте Агрегация MongoDB онлайн: https://riptutorial.com/ru/meteor/topic/4199/агрегация-mongodb

глава 9: Аккаунты пользователей Meteor

Examples

Пакет счетов Meteor

У вас есть несколько вариантов, когда дело доходит до входа в Meteor. Самый распространенный метод - использование `accounts` для Meteor.

Счета-пароль

Если вы хотите, чтобы пользователи могли создавать и регистрироваться на вашем сайте, вы можете использовать `accounts-password` .

Установите пакет, используя `meteor add accounts-password` .

Чтобы создать пользователя, вам необходимо использовать `Accounts.createUser(options, [callback])`

`options` должны быть объектом со следующими свойствами:

- `username` : `username` пользователя пользователя в виде строки.
- `email` : `email` пользователя в виде строки.
- `password` : `password` пользователя (не зашифрованный) как строка.
- `profile` : дополнительные данные пользователя как объект. Это может быть, например, имя пользователя и фамилия пользователя. `profile` является обязательным.

Обратный вызов возвращает 1 переменную, если есть ошибка, которая является объектом Meteor.Error.

Вам нужно только использовать `username` или `email` , чтобы вы могли создать пользователя с именем пользователя, но не с адресом электронной почты, и наоборот. Вы также можете использовать оба варианта.

Он возвращает вновь созданный идентификатор пользователя, если все прошло правильно.

Таким образом, вы можете, например, использовать это:

```
// server side
var id = Accounts.createUser({
    username: "JohnDoe",
    email: "JohnDoe@gmail.com",
```

```
    password: "TheRealJohn123",
    profile: {
        firstName: "John",
        lastName: "Doe"
    }
}, function(err) {
    console.log(err.reason);
});
```

Он также автоматически войдет в систему, если пользователь будет успешно создан.

Это создающая часть. Для входа в систему вам необходимо использовать `Meteor.loginWithPassword(identifier, password, [callback])` на стороне клиента.

`identifier` - это `username` , `email` или `userId` в виде строки от пользователя. `password` - это (не зашифрованный) `password` пользователя.

Обратный вызов возвращает одну переменную, если есть ошибка, которая является объектом Meteor.Error.

Пример:

```
// client side
Meteor.loginWithPassword("JohnDoe", "TheRealJohn123", function(err) {
    console.log(err.reason);
});
```

И это для базового создания учетных записей и входа в систему.

Доступ к пользовательским данным

Вы можете проверить на стороне клиента, если пользователь вошел в систему, вызвав `Meteor.userId()` который вернет свой `userId` если они вошли в систему, и `undefined` если они не вошли в систему.

Вы можете получить некоторую информацию от `Meteor.user()` . Он будет возвращен undefined, если пользователь не войдет в систему, и некоторые данные пользователя, если они есть. По умолчанию он не даст вам никаких паролей, по умолчанию он покажет userId пользователя, имя пользователя и объект профиля.

Если вы хотите проверить, зарегистрирован ли пользователь на странице, вы также можете использовать помощник `currentUser` . Он вернет содержимое `Meteor.user()` . Пример:

```
{{#if currentUser}}
    <h1>Hello there, {{currentUser.username}}!</h1>
{{else}}
    <h1>Please log in.</h1>
{{/if}}
```

Другие функции учетной записи

Существуют и другие функции, которые работают для каждого пакета учетных записей.

Вы можете выйти из системы с помощью `Meteor.logout()`

Не используйте поле профиля по умолчанию

Существует соблазн существующего поля, называемого `profile` который добавляется по умолчанию, когда регистрируется новый пользователь. Это поле исторически предназначалось для использования в качестве царапины для пользовательских данных - возможно, их аватар изображения, имя, вводный текст и т. Д. Из-за этого **поле `profile` для каждого пользователя автоматически записывается этим пользователем с клиента** . Он также автоматически публикуется клиенту для этого конкретного пользователя.

Оказывается, что наличие поля, доступного по умолчанию, не делая этого супер очевидного, может быть не лучшей идеей. Есть много историй о новых разработчиках Meteor, хранящих такие поля, как `isAdmin` on `profile` ..., а затем злоумышленник может легко установить это значение true, когда захочет, сделав себя администратором. Даже если вас это не беспокоит, не рекомендуется позволять злонамеренным пользователям хранить произвольные объемы данных в вашей базе данных.

Вместо того, чтобы иметь дело со спецификой этого поля, может быть полезно просто полностью игнорировать его существование. Это можно сделать безопасно, если вы отказываетесь от всех писем от клиента:

```
// Deny all client-side updates to user documents
Meteor.users.deny({
  update() { return true; }
});
```

Даже игнорируя последствия для профиля безопасности, не рекомендуется включать все пользовательские данные вашего приложения в одно поле. Протокол передачи данных Метеор не делает глубоко вложенных различий полей, поэтому неплохо сгладить ваши объекты во многих областях верхнего уровня документа.

Прочитайте Аккаунты пользователей Meteor онлайн:
https://riptutorial.com/ru/meteor/topic/6219/аккаунты-пользователей-meteor

глава 10: активы

Examples

Доступ к активам на сервере

Статические активы сервера должны быть помещены в `private` каталог.

Текстовые файлы

Доступ к текстовым файлам можно получить с помощью **`Assets.getText(assetPath, [asyncCallback])`** . Например, следующий файл JSON называется `my_text_asset.json` и находится в `private` каталоге:

```
{
    "title": "Meteor Assets",
    "type": "object",
    "users": [{
        "firstName": "John",
        "lastName": "Doe"
    }, {
        "firstName": "Jane",
        "lastName": "Doe"
    }, {
        "firstName": "Matthias",
        "lastName": "Eckhart"
    }]
}
```

Вы можете получить доступ к этому файлу на сервере, используя следующий код:

```
var myTextAsset = Assets.getText('my_text_asset.json');
var myJSON = JSON.parse(myTextAsset);
console.log(myJSON.title); // prints 'Meteor Assets' in the server's console
```

Двоичные файлы

Если вы хотите получить доступ к активам на сервере в виде двоичного кода EJSON, используйте метод **`Assets.getBinary(assetPath, [asyncCallback])`** . Вот пример кода для доступа к изображению с именем `my_image.png` который находится в каталоге `private/img` :

```
var myBinaryAsset = Assets.getBinary('img/my_image.png');
```

Прочитайте активы онлайн: https://riptutorial.com/ru/meteor/topic/3379/активы

глава 11: Горизонтальное масштабирование

Examples

Развертывание приложения с разделенной базой данных (MONGO_URL)

Вам нужно будет отделить свой прикладной уровень от уровня вашей базы данных, а это означает указание MONGO_URL. Это означает, что вы запускаете приложение через команду bundle, распаковываете его, устанавливаете переменные среды и затем запускаете проект в качестве приложения-узла. Вот как...

```
#make sure you're running the node v0.10.21 or later
npm cache clean -f
npm install -g n
sudo n 0.10.21

# bundle the app
mkdir myapp
cd myapp
git clone http://github.com/myaccount/myapp
meteor bundle --directory ../deployPath
cd ../deployPath

# make sure fibers is installed, as per the README
export MONGO_URL='mongodb://127.0.0.1:27017/mydatabase'
export PORT='3000'
export ROOT_URL='http://myapp.com'

# run the site
node main.js
```

Конфигурация набора реплик

Затем перейдите в оболочку mongo и запустите набор реплик, например:

```
mongo

> rs.initiate()
PRIMARY> rs.add("mongo-a")
PRIMARY> rs.add("mongo-b")
PRIMARY> rs.add("mongo-c")
PRIMARY> rs.setReadPref('secondaryPreferred')
```

Настройка набора реплик для использования Oplogging

Для набора реплик потребуется пользователь oplog для доступа к базе данных.

```
mongo

PRIMARY> use admin
PRIMARY>
db.addUser({user:"oplogger",pwd:"YOUR_PASSWORD",roles:[],otherDBRoles:{local:["read"]}});
PRIMARY> show users
```

Сценарий Upstart Oplog

Ваш скрипт upstart необходимо будет изменить, чтобы использовать несколько IP-адресов набора реплик.

```
start on started mountall
stop on shutdown

respawn
respawn limit 99 5

script
    # our example assumes you're using a replica set and/or oplog integreation
    export MONGO_URL='mongodb://mongo-a:27017,mongo-b:27017,mongo-c:27017/meteor'

    # here we configure our OPLOG URL
    export MONGO_OPLOG_URL='mongodb://oplogger:YOUR_PASSWORD@mongo-a:27017,mongo-
b:27017,mongo-c:27017/local?authSource=admin'

    # root_url and port are the other two important environment variables to set
    export ROOT_URL='http://myapp.mydomain.com'
    export PORT='80'

    exec /usr/local/bin/node /var/www/production/main.js >> /var/log/node.log 2>&1
end script
```

Sharding

Oplog Tailing on Sharded Mongo

Прочитайте Горизонтальное масштабирование онлайн:
https://riptutorial.com/ru/meteor/topic/3706/горизонтальное-масштабирование

глава 12: Доступ к машинам сборки Meteor из Windows

замечания

В Mac и Linux инструмент командной строки `meteor` предполагает, что инструмент командной строки `ssh` , используемый для безопасного подключения к другим компьютерам, всегда присутствует. В Windows этот инструмент необходимо установить. Ниже перечислены два варианта настройки и использования.

Examples

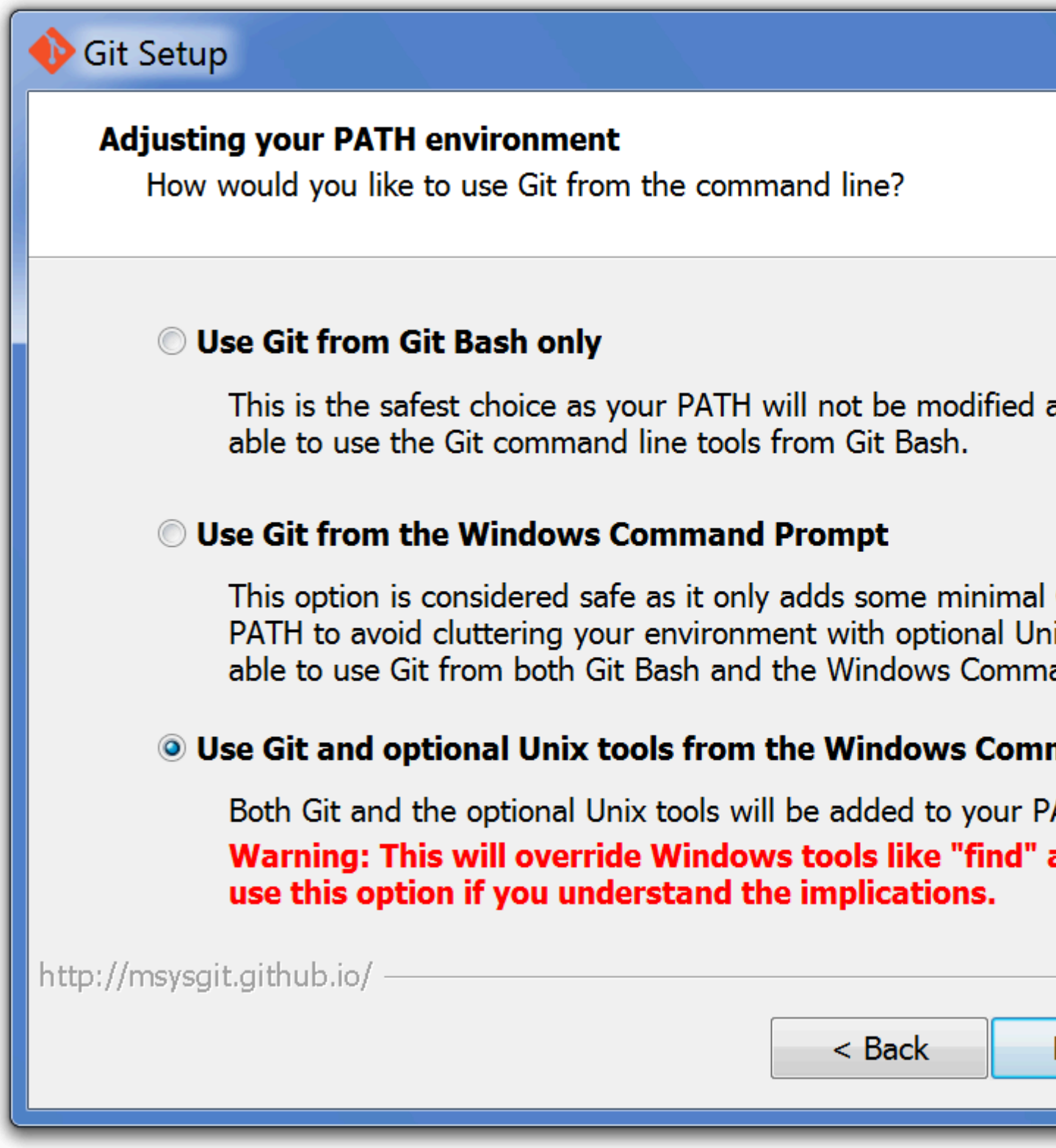
Использование PuTTY (Advanced)

Если вы не хотите добавлять команды Unix в свою PATH в Windows, вы можете загрузить автономный клиент SSH, такой как PuTTY. Загрузите PuTTY здесь , а затем следуйте инструкциям ниже, чтобы получить машину сборки.

1. Call `meteor admin get-machine <os-architecture> --json`
2. Скопируйте и сохраните закрытый ключ из возвращаемых данных JSON
3. Следуйте указаниям здесь, чтобы преобразовать закрытый ключ в формат, который принимает PuTTY.
4. Введите имя хоста, имя пользователя и закрытый ключ в PuTTY, и вы готовы к работе!

Использование Cygwin (инструменты Unix для Windows)

Самый простой способ встать и запустить - установить Git для Windows с этой страницы загрузки и выбрать «Использовать Git и дополнительные инструменты Unix из командной строки Windows», как показано на скриншоте ниже.



После этого, `meteor admin get-machine <os-architecture>` будет работать точно так же, как на Linux и Mac. Имейте в виду, что вам может понадобиться перезагрузить терминал, чтобы получить новые команды.

Прочитайте Доступ к машинам сборки Meteor из Windows онлайн:
https://riptutorial.com/ru/meteor/topic/518/доступ-к-машинам-сборки-meteor-из-windows

глава 13: Загрузка файлов

замечания

Пакет CollectionFS был отложен и прекращен автором; однако, поскольку нет альтернативного пакета в Атмосфере или экосистеме Метеор для использования функции Mongo GridFS, и код все еще работает отлично; мы рекомендуем не удалять этот пример из документации StackOverflow до тех пор, пока какое-либо другое решение GridFS не будет задокументировано как замена.

Дополнительные исследования
Загрузка файлов и изображений
Файл сохранения файла Dario
Образец загрузки файла Micha Roon
Пакет загрузки файлов EventedMind

Examples

Сервер / Клиент

Загрузка файлов может быть легкой или очень сложной, в зависимости от того, что вы хотите делать. В общем, передача самого файла не так уж трудна. Но есть множество краевых случаев вокруг вложений, двоичных файлов и тому подобного. И реальная точка прилипания - горизонтальное масштабирование и создание решения, которое работает, когда сервер клонируется второй, третий и n-й раз.

Начнем с базовой модели загрузки сервера / клиента. Начнем с добавления элемента ввода файла в объектную модель документа.

```
<template name="example">
  <input type=file />
</template>
```

Затем присоедините событие к элементу ввода внутри вашего контроллера и вызовите локальный метод Meteor `` startFileTransfer '', чтобы инициировать передачу.

```
// client/example.js
Template.example.events({
  'change input': function(ev) {
    _.each(ev.srcElement.files, function(file) {
      Meteor.startFileTransfer(file, file.name);
    });
  }
});
```

```
// client/save.js
/**
 * @blob (https://developer.mozilla.org/en-US/docs/DOM/Blob)
 * @name the file's name
 * @type the file's type: binary, text (https://developer.mozilla.org/en-
US/docs/DOM/FileReader#Methods)
 *
 * TODO Support other encodings: https://developer.mozilla.org/en-
US/docs/DOM/FileReader#Methods
 * ArrayBuffer / DataURL (base64)
 */
Meteor.startFileTransfer = function(blob, name, path, type, callback) {
  var fileReader = new FileReader(),
    method, encoding = 'binary', type = type || 'binary';
  switch (type) {
    case 'text':
      // TODO Is this needed? If we're uploading content from file, yes, but if it's from an
input/textarea I think not...
      method = 'readAsText';
      encoding = 'utf8';
      break;
    case 'binary':
      method = 'readAsBinaryString';
      encoding = 'binary';
      break;
    default:
      method = 'readAsBinaryString';
      encoding = 'binary';
      break;
  }
  fileReader.onload = function(file) {
    Meteor.call('saveFileToDisk', file.srcElement.result, name, path, encoding, callback);
  }
  fileReader[method](blob);
}
```

Затем клиент вызовет метод сервера saveFileToDisk, который выполняет фактическую передачу и помещает все на диск.

```
//
/**
 * TODO support other encodings:
 * http://stackoverflow.com/questions/7329128/how-to-write-binary-data-to-a-file-using-node-js
 */
Meteor.methods({
  saveFileToDisk: function(blob, name, path, encoding) {
    var path = cleanPath(path), fs = __meteor_bootstrap__.require('fs'),
      name = cleanName(name || 'file'), encoding = encoding || 'binary',
      chroot = Meteor.chroot || 'public';
    // Clean up the path. Remove any initial and final '/' -we prefix them-,
    // any sort of attempt to go to the parent directory '..' and any empty directories in
    // between '/////' - which may happen after removing '..'
    path = chroot + (path ? '/' + path + '/' : '/');

    // TODO Add file existance checks, etc...
    fs.writeFile(path + name, blob, encoding, function(err) {
      if (err) {
        throw (new Meteor.Error(500, 'Failed to save file.', err));
      } else {
```

```
        console.log('The file ' + name + ' (' + encoding + ') was saved to ' + path);
      }
    });

    function cleanPath(str) {
      if (str) {
        return str.replace(/\.\./g,'').replace(/\/+/g,'').
          replace(/^\/+/,'').replace(/\/+$/,'');
      }
    }
    function cleanName(str) {
      return str.replace(/\.\./g,'').replace(/\//g,'');
    }
  }
});
```

Это своего рода подход с голыми костями, и это оставляет желать лучшего. Возможно, это полезно для загрузки CSV-файла или чего-то еще, но это все.

Dropzone (с железом: маршрутизатор)

Если нам нужно что-то более отполированное, с интегрированным интерфейсом Dropzone и конечной точкой REST, нам нужно будет начать добавлять пользовательские маршруты и пакеты REST с помощью помощников пользовательского интерфейса.

Начнем с импорта Iron Router и Dropzone.

```
meteor add iron:router
meteor add awatson1978:dropzone
```

И настройте URL-адрес загружаемого URL-адреса, указанный в помощнике dropzone.

```
Router.map(function () {
    this.route('uploads', {
      where: 'server',
      action: function () {
        var fs = Npm.require('fs');
        var path = Npm.require('path');
        var self = this;

        ROOT_APP_PATH = fs.realpathSync('.');

        // dropzone.js stores the uploaded file in the /tmp directory, which we access
        fs.readFile(self.request.files.file.path, function (err, data) {

          // and then write the file to the uploads directory
          fs.writeFile(ROOT_APP_PATH + "/assets/app/uploads/" +self.request.files.file.name,
data, 'binary', function (error, result) {
            if(error){
              console.error(error);
            }
            if(result){
              console.log('Success! ', result);
            }
          });
```

```
        });
      }
    });
  });
```

Здорово! У нас есть файловый загрузчик с потрясающим интерфейсом и программируемой конечной точкой REST. К сожалению, это не очень хорошо масштабируется.

Filepicker.io

Чтобы масштабировать ситуацию, мы должны прекратить использование локального хранилища на нашем сервере и начать использовать либо специализированную службу хранения файлов, либо реализовать горизонтальный уровень хранения. Самый простой способ начать работу с масштабируемым хранилищем файлов - использовать такое решение, как Filepicker.io, которое поддерживает S3, Azure, Rackspace и Dropbox. loadpicker был популярным Unipackage Filerpicker на некоторое время.

```
meteor add mrt:filepicker
```

Шаблон Filepicker отличается от других решений, потому что это действительно о сторонней интеграции. Начните с добавления ввода filepicker, который, как вы увидите, в значительной степени зависит от атрибутов data- *, что довольно необычно для приложений Meteor.

```
<input type="filepicker"
  id="filepickerAttachment"
  data-fp-button-class="btn filepickerAttachment"
  data-fp-button-text="Add image"
  data-fp-mimetypes="image/*"
  data-fp-container="modal"
  data-fp-maxsize="5000000"
  data-fp-services="COMPUTER,IMAGE_SEARCH,URL,DROPBOX,GITHUB,GOOGLE_DRIVE,GMAIL">
```

Вы также захотите установить ключ API, построить виджет filepicker, запустить его и наблюдать за его результатами.

```
if(Meteor.isClient){
  Meteor.startup(function() {
    filepicker.setKey("YourFilepickerApiKey");
  });
  Template.yourTemplate.rendered = function(){
    filepicker.constructWidget($("#filepickerAttachment"));
  }
  Template.yourTemplate.events({
  'change #filepickerAttachment': function (evt) {
    console.log("Event: ", evt, evt.fpfile, "Generated image url:", evt.fpfile.url);
  });
});
```

CollectionFS

Однако, если вы действительно серьезно относитесь к хранилищу и хотите хранить миллионы изображений, вам понадобится использовать инфраструктуру GridFS Mongo и создать себе уровень хранения. Для этого вам понадобится отличная подсистема CollectionFS.

Начните с добавления необходимых пакетов.

```
meteor add cfs:standard-packages
meteor add cfs:filesystem
```

И добавление элемента загрузки файла в вашу объектную модель.

```
<template name="yourTemplate">
    <input class="your-upload-class" type="file">
</template>
```

Затем добавьте контроллер событий на клиенте.

```
Template.yourTemplate.events({
    'change .your-upload-class': function(event, template) {
        FS.Utility.eachFile(event, function(file) {
            var yourFile = new FS.File(file);
            yourFile.creatorId = Meteor.userId(); // add custom data
            YourFileCollection.insert(yourFile, function (err, fileObj) {
                if (!err) {
                   // do callback stuff
                }
            });
        });
    }
});
```

И определите свои коллекции на своем сервере:

```
YourFileCollection = new FS.Collection("yourFileCollection", {
    stores: [new FS.Store.FileSystem("yourFileCollection", {path: "~/meteor_uploads"})]
});
YourFileCollection.allow({
    insert: function (userId, doc) {
        return !!userId;
    },
    update: function (userId, doc) {
        return doc.creatorId == userId
    },
    download: function (userId, doc) {
        return doc.creatorId == userId
    }
});
```

Благодаря Raz за этот отличный пример. Для получения более подробной информации о том, что может сделать все CollectionFS, вы захотите ознакомиться с полной сборкой CollectionFS.

Загрузка серверов

Следующие сценарии предназначены для загрузки файла из файловой системы сервера на сервер. В основном для файлов конфигурации и файловых шаблонов.

```
//https://forums.meteor.com/t/read-file-from-the-public-folder/4910/5

// Asynchronous Method.
Meteor.startup(function () {
    console.log('starting up');

    var fs = Npm.require('fs');
    // file originally saved as public/data/taxa.csv
    fs.readFile(process.cwd() + '/../web.browser/app/data/taxa.csv', 'utf8', function (err,
data) {
        if (err) {
            console.log('Error: ' + err);
            return;
        }

        data = JSON.parse(data);
        console.log(data);
    });
});


// Synchronous Method.
Meteor.startup(function () {
    var fs = Npm.require('fs');
    // file originally saved as public/data/taxa.csv
    var data = fs.readFileSync(process.cwd() + '/../web.browser/app/data/taxa.csv', 'utf8');

    if (Icd10.find().count() === 0) {
        Icd10.insert({
            date:  new Date(),
            data:  JSON.parse(data)
        });
    }
});


Meteor.methods({
  parseCsvFile:function (){
    console.log('parseCsvFile');

    var fs = Npm.require('fs');
    // file originally saved as public/data/taxa.csv
    var data = fs.readFileSync(process.cwd() + '/../web.browser/app/data/taxa.csv', 'utf8');
    console.log('data', data);
  }
});
```

Прочитайте Загрузка файлов онлайн: https://riptutorial.com/ru/meteor/topic/3119/загрузка-файлов

глава 14: Извлечение данных из Meteor.call

Examples

Основы Meteor.call

```
Meteor.call(name, [arg1, arg2...], [asyncCallback])
```

(1) имя String
(2) Имя метода для вызова
(3) arg1, arg2 ... Объект, поддерживающий EJSON [Дополнительно]
(4) Функция asyncCallback [Необязательно]

С одной стороны, вы можете делать: (через **переменную сеанса** или через **ReactiveVar**)

```
    var syncCall = Meteor.call("mymethod") // Sync call
```

Это означает, что если вы сделаете что-то вроде этого, серверная сторона вы будете делать:

```
    Meteor.methods({
        mymethod: function() {
            let asyncToSync =  Meteor.wrapAsync(asynchronousCall);
            // do something with the result;
            return  asyncToSync;
        }
    });
```

С другой стороны, иногда вы захотите сохранить его в результате обратного вызова?

Сторона клиента :

```
Meteor.call("mymethod", argumentObjectorString, function (error, result) {
    if (error) Session.set("result", error);
    else Session.set("result",result);
}
Session.get("result") -> will contain the result or the error;

//Session variable come with a tracker that trigger whenever a new value is set to the session
variable. \ same behavior using ReactiveVar
```

Серверная сторона

```
Meteor.methods({
    mymethod: function(ObjectorString) {
        if (true) {
            return true;
        } else {
```

```
            throw new Meteor.Error("TitleOfError", "ReasonAndMessageOfError"); // This will
and up in the error parameter of the Meteor.call
        }
    }
});
```

Цель здесь - показать, что Meteor предлагает различный способ связи между Клиентом и Сервером.

Использование переменной сеанса

Серверная сторона

```
Meteor.methods({
  getData() {
    return 'Hello, world!';
  }
});
```

Сторона клиента

```
<template name="someData">
  {{#if someData}}
    <p>{{someData}}</p>
  {{else}}
    <p>Loading...</p>
  {{/if}}
</template>
```

```
Template.someData.onCreated(function() {
  Meteor.call('getData', function(err, res) {
    Session.set('someData', res);
  });
});

Template.someData.helpers({
  someData: function() {
    return Session.get('someData');
  }
});
```

Использование ReactiveVar

Серверная сторона

```
Meteor.methods({
  getData() {
    return 'Hello, world!';
  }
});
```

Сторона клиента

```
<template name="someData">
  {{#if someData}}
    <p>{{someData}}</p>
  {{else}}
    <p>Loading...</p>
  {{/if}}
</template>
```

```
Template.someData.onCreated(function() {

  this.someData = new ReactiveVar();

  Meteor.call('getData', (err, res) => {
    this.someData.set(res);
  });
});

Template.someData.helpers({
  someData: function() {
    return Template.instance().someData.get();
  }
});
```

требуется пакет `reactive-var` . Чтобы добавить его, запустите `meteor add reactive-var` .

Прочитайте Извлечение данных из Meteor.call онлайн:
https://riptutorial.com/ru/meteor/topic/3068/извлечение-данных-из-meteor-call

глава 15: Издательский выпуск

замечания

Публикация выпуска трека на самом деле довольно проста, если вы понимаете, a) что команда publish-release требует файла .json в качестве параметра и b) как выглядит этот файл. Это определенно самый большой барьер в начале работы, потому что он практически не документирован нигде.

Просто имейте в виду, что каждый пакет в выпуске должен быть опубликован и в Atmosphere. Файл .meteor / versions приложения является особенно хорошим местом для поиска всех необходимых пакетов и версий, которые должны войти в релиз.

После этого нужно выяснить, что вы готовы поддержать, что хотите включить и т. Д. Вот частичная диаграмма Венна о том, что сейчас работает в клинической версии; и должен дать вам общее представление о том, как мы собираемся принять процесс принятия решений о включении.

Для более подробной информации см. Раздел на форумах Meteor:
https://forums.meteor.com/t/custom-meteor-release/13736/6

Examples

Основное использование

Идея состоит в том, что разработчик дистрибутивов хочет запустить что-то вроде следующей команды:

```
meteor publish-release clinical.meteor.rc6.json
```

Затем это позволит пользователям дистрибутива запускать это:

```
meteor run --release clinical:METEOR@1.1.3-rc6
```

Выпуск манифеста

Выпуск релиза аналогичен файлу NPM `package.json` , поскольку основной проблемой является список пакетов Atmosphere и предоставление нескольких метаданных об этом списке пакетов. Основной формат выглядит следующим образом:

```
{
  "track":"distroname:METEOR",
  "version":"x.y.z",
```

```
  "recommended": false,
  "tool": "distroname:meteor-tool@x.y.z",
  "description": "Description of the Distro",
  "packages": {
    "accounts-base":"1.2.0",
    "accounts-password":"1.1.1",
    ...
  }
}
```

Настройка инструмента Метеор

Если вам необходимо расширить инструмент метеоритов или командную строку, вам нужно будет создать и опубликовать собственный пакет метеорных инструментов. Документация Ронена лучше всего подходит для этого процесса:

http://practicalmeteor.com/using-meteor-publish-release-to-extend-the-meteor-command-line-tool/1

Легко получить команду meteor helloworld, но после этого я почувствовал, что проще просто создать отдельное приложение для тестирования команд. Именно так появился StarryNight. Это что-то вроде промежуточной площадки и блокнот для команд, прежде чем пытаться вставить их в версию метеорного инструмента.

Извлечение манифеста выпуска из .meteor / версий

StarryNight содержит небольшую утилиту, которая анализирует файл `.meteor/versions` и преобразует его в манифест Release.

```
npm install -g starrynight
cd myapp
starrynight generate-release-json
```

Если вы не хотите использовать StarryNight, просто скопируйте содержимое вашего `.meteor/versions` в поле `packages` вашего файла манифеста. Обязательно конвертируйте в синтаксис JSON и добавьте двоеточия и кавычки.

Отображение манифеста выпуска для определенного выпуска

```
meteor show --ejson METEOR@1.2.1
```

Публикация выпуска с Checkout

```
meteor publish-release --from-checkout
```

Получение последних коммитов для каждого пакета в выпуске

При создании пользовательской дорожки выпуска обычно хранятся пакеты в каталоге `/packages` виде подмодулей git. Следующая команда позволяет вам одновременно получать все последние коммиты для подмодулей в каталоге `/packages` .

```
git submodule foreach git pull origin master
```

Прочитайте Издательский выпуск онлайн: https://riptutorial.com/ru/meteor/topic/4201/издательский-выпуск

глава 16: Инструменты разработки

Examples

Интегрированные среды разработки

Обычно разработка начинается с редактора или интегрированной среды разработки. Известно, что следующие IDE поддерживают Meteor:

- Atom - Javascript IDE, которая может полностью использовать изоморфную структуру JavaScript JavaScript. Если вы хотите, чтобы иметь возможность взломать ваш редактор, это тот, который нужно выбрать.
- Cloud9 - новейшее предложение Cloud Development, поддерживающее Meteor, с учебником.
- MeteorDevTools - расширение Chrome для Blaze, DDP и Minimongo.
- Sublime - легкий и популярный текстовый редактор.
- WebStorm - наиболее полнофункциональная IDE, доступная в настоящее время для Meteor.

Инструменты базы данных

Как только вы пройдете свое приложение «Hello World», вам нужно будет начать обращать внимание на свои схемы сбора и документов и понадобятся некоторые инструменты для управления вашей базой данных.

- Robomongo - давняя общественность, любимая для управления Монго. Настоятельно рекомендуется.
- Генератор JSON - бесценная утилита для создания выборочных наборов данных.
- Страница предпочтений MacOSX Mongo - Предпочтения GUI для MacOSX.
- MongoHub - еще один Mongo GUI, похожий на RoboMongo. Только MacOSX.
- Mongo3 - один из немногих инструментов управления кластерами. Возможность визуализации наборов репликации. Единственный недостаток - это встроенный в Ruby.
- Служба мониторинга Mongo - Когда вы готовы принести что-то в производство, MMS бесценен. Теперь известен как MongoDB Atlas.
- Mongo Express - веб-интерфейс администратора MongoDB, написанный с помощью Node.js и выражающий

Утилиты удаленного взаимодействия для распределенных разработчиков

Разработка приложений Meteor обычно означает разработку многоклиентской

реактивности, которая требует инструментов совместной работы. Следующие инструменты оказались популярными в сообществе Meteor.

- Google Hangouts - видеоконференции и чат.
- Zenhub.io - Kanban для GitHub.
- InVision - совместное каркасное и прототипирование.
- Встреча с героем - планирование коллективных встреч.
- Hackpad - Совместное редактирование документов.
- Slack - совместные трассировки отслеживания проектов.
- MadEye - совместный веб-редактор.
- Screenhero - совместное совместное использование экрана.
- Proto.io - каркас и прототипирование.
- Доска HuBoard - Kanban для GitHub.
- Zapier - лучшие приложения. Все вместе.
- Teamwork.com - традиционное руководство проектом и диаграммы gannt.
- Sprint.ly - Больше советов канбан и планирование спринта, которые работают с GitHub.
- LucidChart - альтернатива Online Visio.
- Waffle.io - альтернатива Trello / ZenHub, которая интегрируется с GitHub.

Клиенты REST

Если вы хотите интегрировать Meteor с внешним API, вполне вероятно, что он будет отображаться как интерфейс REST. Мы используем следующие приложения Chrome для тестирования API REST.

- Почтальон
- Клиент DHC Rest

Онлайн-инструменты:

- Hurl.it
- RequestBin

Debuggers

Большая часть отладки происходит в терминале или в инструментах разработки Chrome или Safari, которые достаточно сложны для 99% ваших потребностей. Однако, если вы хотите отлаживать Firefox или нуждаться в дополнительной функции отладки сервера, есть несколько дополнительных утилит, которые вы можете использовать.

- Firefox - Firebug
- Узел-инспектор
- Метеорные игрушки или `meteor add meteortoys:allthings` добавлением прямых `meteor add meteortoys:allthings`

Мобильное кодирование на iOS

Texttastic Code Editor - Редактор кода с подсветкой синтаксиса для устройств iOS.

Рабочая копия - клонирование хранилищ Github на iPad и код в пути.

CodeHub - просмотр и поддержка репозиториев GitHub. Инструмент управления.

iOctocat - Социальная утилита для следующих проектов Github.

iMockups для iPad - каркасы и макеты. Поддерживает каркасы для настольных компьютеров и мобильных устройств.

Blueprint - каркас iOS и макеты. В первую очередь для разработки iOS, но несколько пригодных для использования в веб-приложениях.

JSON Designer - Архитектура архитектуры данных и схемы данных.

Прочитайте Инструменты разработки онлайн: https://riptutorial.com/ru/meteor/topic/4200/инструменты-разработки

глава 17: Интеграция сторонних API

Examples

Основной HTTP-вызов

Концептуально интегрировать сторонние API REST можно так же просто, как добавить пакет `http` и сделать вызов внешней конечной точке.

```
meteor add http
```

```
HTTP.get('http://foo.net/api/bar/');
```

Создайте пакет для вашего API-интерфейса

Однако основные HTTP-вызовы не обеспечивают повторного использования кода. И они могут запутаться со всеми другими функциями, которые вы пытаетесь реализовать. По этим причинам обычно используется оболочка API.

```
Foo = {
  identify: function(input){
    return Http.get('http://foo.net/api/identify/' + input);
  },
  record_action_on_item: function(firstInput, secondInput){
    return Http.put('http://foo.net/api/record_action_on_item/' + firstInput + '&' +
secondInput);
  }
}
```

Meteor поддерживает Http.get (), Http.post (), Http.put () и т. Д., Поэтому это, несомненно, лучший способ вызвать ваш REST API. http://docs.meteor.com/#http_get

Если API является частым и подробным, вы можете получать несколько пакетов; в этом случае вам нужно будет их собрать. Это большая проблема. Если вы считаете, что API возвращает несколько пакетов, вероятно, вам захочется использовать модуль «request» npm на сервере. Вы захотите использовать `Npm.require('request')` .
https://github.com/mikeal/request

Создайте пакет Атмосферы для вашего API-интерфейса

После создания обертки API, вероятно, вам захочется создать пакет Atmosphere для его распространения и обмена им между приложениями. Файлы вашего пакета, вероятно, будут выглядеть примерно так.

```
packages/foo-api-wrapper/package.js
```

```
packages/foo-api-wrapper/readme.md
packages/foo-api-wrapper/foo.api.wrapper.js
```

В частности, ваш файл `foo-api-wrapper/package.js` захочет выглядеть примерно так:

```
Package.describe({
  summary: "Atmosphere package that impliments the Foo API.",
  name: "myaccount:foo",
  version: '0.0.1'
});

Package.on_use(function (api) {
    api.export('Foo');
    api.addFiles('foo.api.wrapper.js', ["client","server"]);
});
```

И ваш `foo-api-wrapper/foo.api.wrapper.js` должен содержать объект оболочки `Foo` API.

Включите пакет API в приложение

На данный момент вы все еще строите свой пакет, поэтому вам нужно добавить пакет в свое приложение:

```
meteor add myaccount:foo
```

И, в конце концов, опубликуйте его в Atmosphere:

```
meteor publish myaccount:foo
```

Использование объекта API Wrapper в вашем приложении

Теперь, когда мы собрали все эти части, теперь вы сможете делать такие вызовы, как из вашего приложения:

```
Foo.identify('John');
Foo.record_action_on_item('view', "HackerNews');
```

Очевидно, что вам нужно настроить имена функций, аргументы, URL-адреса и т. П., Чтобы создать правильный синтаксис для API.

Прочитайте Интеграция сторонних API онлайн: https://riptutorial.com/ru/meteor/topic/3118/интеграция-сторонних-api

глава 18: Использование Meteor с прокси-сервером

Examples

Использование `HTTP [S] _PROXY` env var

На этой странице описывается использование средства командной строки Meteor (например, при загрузке пакетов, развертывании вашего приложения и т. Д.) За прокси-сервером.

Как и многие другие программные средства командной строки, инструмент Meteor считывает конфигурацию прокси-сервера из переменных среды `HTTP_PROXY` и `HTTPS_PROXY` (также работают варианты нижнего регистра). Примеры запуска Meteor за прокси:

- на Linux или Mac OS X

```
export HTTP_PROXY=http://user:password@1.2.3.4:5678
export HTTPS_PROXY=http://user:password@1.2.3.4:5678
meteor update
```

- в Windows

```
SET HTTP_PROXY=http://user:password@1.2.3.4:5678
SET HTTPS_PROXY=http://user:password@1.2.3.4:5678
meteor update
```

Настройка уровня прокси

- Разверните приложение Meteor для Ubuntu с помощью прокси-сервера Nginx
- Как создать сертификат SSL для Nginx для Ubuntu 14
- Как развернуть приложение Meteor JS на Ubuntu с Nginx
- Как установить сертификат SSL из центра сертификации
- ИмяCheap SSL-сертификаты

Прочитайте Использование Meteor с прокси-сервером онлайн:
https://riptutorial.com/ru/meteor/topic/517/использование-meteor-с-прокси-сервером

глава 19: Использование полимера с метеор

Examples

Использование дифференциала: вулканизация

В корне вашего проекта убедитесь, что Bower установлен (`npm install -g bower`) и запускается `bower init` . Это создаст файл `bower.json` в каталоге вашего проекта.

Создайте новый файл с именем `.bowerrc` в корневой каталог. Он должен содержать следующее:

```
{
  "directory": "public/bower_components"
}
```

Это позволяет Bower знать, что он должен сохранять компоненты в папке `bower_components` в `bower_components` каталоге вашего приложения.

Теперь добавьте компоненты Polymer, которые вы хотите использовать с вашим приложением.

В корневой каталог вашего приложения bower - установите каждый компонент, который вы хотите использовать.

```
bower install --save PolymerElements/paper-button#^1.0.0 PolymerElements/paper-checkbox#^1.0.0
```

Добавьте Vulcanize в свой проект

```
Meteor add differential:vulcanize
```

Создайте новый файл config.vulcanize в корне вашего проекта. Он должен содержать следующее:

```
{
    "polyfill": "/bower_components/webcomponentsjs/webcomponents.min.js",
    "useShadowDom": true, // optional, defaults to shady dom (polymer default)
    "imports": [
        "/bower_components/paper-button/paper-button.html",
        "/bower_components/paper-checkbox/paper-checkbox.html"
    ]
}
```

`"imports"` должен содержать список каждого компонента, который вы будете использовать

в своем приложении.

Теперь вы можете использовать компоненты, импортированные в шаблоны Blaze, как и любой другой элемент:

```
<template name="example">
    <div>
        this is a material design button: <paper-button></paper-button>
        this is a material design checkbox: <paper-checkbox></paper-checkbox>
    </div>
</template>
```

Прочитайте Использование полимера с метеор онлайн:
https://riptutorial.com/ru/meteor/topic/4598/использование-полимера-с-метеор

глава 20: Использование частных метеорных пакетов на кодах

замечания

Обратите внимание, что мы не обсуждали, как использовать и разрабатывать локальные пакеты. Есть несколько способов, я предлагаю использовать `PACKAGE_DIRS` окружения `PACKAGE_DIRS` описанную Дэвидом Уэлдоном на его веб-сайте .

Examples

Установка MGP

Мы используем пакет Dispatches Great Meteor Github Packages (mgp) :

```
npm install --save mgp
```

Затем добавьте следующую команду в свои скрипты `package.json` :

```
"mgp": "mgp"
```

Создайте файл с именем `git-packages.json` в корне вашего проекта. Добавьте конфигурацию для каждого (частного) пакета Meteor Github, от которого зависит ваш проект:

```
{
  "my:yet-another-private-package": {
    "git": "git@github.com:my/private-packages.git",
    "branch": "dev"
  }
}
```

Более подробную информацию о настройке ваших частных пакетов можно найти в проектах Github repo.

Настройка Codeship для установки частных пакетов Github

Добавьте команду команды Codeship в следующую команду:

```
meteor npm run mgp
```

Теперь нам нужно предоставить доступ к Codeship этим частным репозиториям. Существует документация по документации Кодекса, подробно описывающая этот процесс,

но вот шаги, которые вы должны предпринять для Github:

- Создайте новую учетную запись Github. Так называемый пользователь машины .
- Удалите ключ развертывания из тестируемого репо. Здесь: https://github.com/YOUR_USERNAME/REPO_UNDER_TEST/settings/keys
- Возьмите открытый ключ SSH из настроек ваших проектов кодов. Где-то здесь: https://codeship.com/projects/PROJECT_NUMBER/configure
- Добавьте этот открытый SSH ключ к ключам SSH вашего компьютера: https://github.com/settings/keys
- Предоставьте этому пользователю доступ к вашим репозиториям, на которые вы ссылаетесь

Он должен быть похож на BitBucket и другие.

Прочитайте Использование частных метеорных пакетов на кодах онлайн: https://riptutorial.com/ru/meteor/topic/6742/использование-частных-метеорных-пакетов-на-кодах

глава 21: логирование

Examples

Базовая регистрация на стороне сервера

Первый шаг к регистрации - просто запустить Meteor из оболочки, и вы получите журналы сервера в командной консоли.

```
meteor
```

Следующий шаг - передать содержимое std_out и std_err в файл журнала, например:

```
meteor > my_app_log.log 2> my_app_err.log
```

Инструменты ведения журнала на стороне клиента

После того, как вы зарегистрируетесь на стороне сервера, пришло время перейти на сторону клиента. Если вы не изучили API-интерфейс консоли, будьте готовы к удовольствию. На самом деле есть все, что вы можете сделать со встроенным API Console, который является родным для каждой установки Chrome и Safari. На самом деле, на самом деле, вы, возможно, не нуждаетесь в Winston или других фреймворках регистрации.

Первое, что вам нужно сделать, это установить средства регистрации на стороне клиента и инструменты для разработчиков. Chrome и Safari поставляются вместе с ними, но Firefox требует расширения Firebug.

Расширение Firebug

Затем вы захотите ознакомиться с документацией консоли API. Следующие два документа являются неоценимыми ресурсами для обучения в консоли.

Инструменты разработчика Chrome

Firebug (клиент)

Расширенные инструменты ведения журнала сервера

После того, как вы запустили ведение журналов на стороне сервера, а также инструменты разработки на стороне клиента, вы можете начать просмотр специальных расширений Meteor, таких как расширение Meteor Chrome DevTools Extension. Это позволяет вам наблюдать за сервером в клиенте! Потому что база данных везде. Как и регистрация.

Расширение Chrome DevTools (сервер)

Ошибка регистрации на лотке базы данных

Следующий пример - от 0,5 до 0,7 дня и иллюстрирует, как регистрировать ошибку, когда база данных еще не заполнила курсор на стороне клиента.

```
Template.landingPage.postsList = function(){
  try{
    return Posts.find();
  }catch(error){
    //color code the error (red)
    console.error(error);
  }
}
```

Информация о регистрации в контексте данных в помощнике шаблона

Следующее использует API ведения журнала Chrome. Если синтаксис `.group()` используется в нескольких шаблонах, он будет графически организовывать журналы консоли из разных шаблонов в иерархическое дерево.

Вы также можете посмотреть, как проверить текущий контекст данных и как укрепить данные.

```
Template.landingPage.getId = function(){
  // using a group block to illustrate function scoping
  console.group('coolFunction');

  // inspect the current data object that landingPage is using
  console.log(this);

  // inspect a specific field of the locally scoped data object
  console.log(JSON.stringify(this._id);

  // close the function scope
  console.groupEnd();
  return this._id;
}
```

Ведение журнала событий и взаимодействия с пользователем

Простой пример использования API регистрации Chrome.

```
Template.landingPage.events({
  'click .selectItemButton':function(){
    // color code and count the user interaction (blue)
    console.count('click .selectItemButton');
  }
});
```

Ведение журнала с помощью переменных уровня журнала

Ведение журнала часто может загромождать консоль, поэтому обычно определяют уровни журналов для контроля того, какая деталь данных регистрируется. Общий шаблон - это указать переменные уровня журнала.

```
var DEBUG = false;
var TRACE = false;
Template.landingPage.events({
  'click .selectItemButton':function(){
    TRACE && console.count('click .selectItemButton');

    Meteor.call('niftyAction', function(errorMessage, result){
        if(errorMessage){
            DEBUG && console.error(errorMessage);
        }
    });
  }
});
```

Отключить регистрацию в производстве

Некоторые команды обнаруживают, что хотят оставить консольные записи журнала в своем коде, но не показывать их на рабочем месте. Они будут переопределять функции ведения журнала, если переменная не установлена (возможно, переменная среды). Кроме того, это может квалифицироваться как функция безопасности в некоторых ситуациях.

```
if (!DEBUG_MODE_ON) {
    console = console || {};
    console.log = function(){};

    console.log = function(){};
    console.error = function(){};
    console.count = function(){};
    console.info = function(){};
}
```

Winston

Если вам нужно что-то более мощное, чем параметры ведения журнала по умолчанию, вы можете посмотреть на такой инструмент, как Winston. Перейдите в Атмосферу и просто найдите один из множества доступных пакетов Winston.

https://atmospherejs.com/?q=winston

Однако будьте осторожны - Winston - это сложный продукт, и, хотя он предоставляет много функциональности, он также добавит уровень сложности вашему приложению.

LOGLEVEL

Следует особо отметить, что сообщество разработало пакет LogLevel. Похоже, что он балансирует между легким и простым в использовании, хорошо работая с конвейером

пакетов Meteor и сохраняя номера строк и имена файлов.

https://atmospherejs.com/practicalmeteor/loglevel

Прочитайте логирование онлайн: https://riptutorial.com/ru/meteor/topic/3376/логирование

глава 22: маршрутизация

Examples

Маршрутизация с помощью Iron Router

Установка Iron Router

От терминала:

```
meteor add iron:router
```

Основная конфигурация

```
Router.configure({
    //Any template in your routes will render to the {{> yield}} you put inside your layout
template
    layoutTemplate: 'layout',
    loadingTemplate: 'loading'
});
```

Рендеринг без данных

```
//this is equal to home page
Router.route('/', function (){
    this.render('home')
});

Router.route('/some-route', function () {
    this.render('template-name');
});
```

Отображать данные и параметры

```
Router.route('/items/:_id', function () {
    this.render('itemPage', {
        data: function() {
            return Items.findOne({_id: this.params._id})
        }
    });
});
```

Отдавать вторичный доход

```
Router.route('/one-route/route', function() {
    //template 'oneTemplate' has {{> yield 'secondary'}} in HTML
    this.render('oneTemplate');

    //this yields to the secondary place
    this.render('anotherTemplate', {
```

```
        to: 'secondary'
    });

    //note that you can write a route  for '/one-route'
    //then another for '/one-route/route' which will function exactly like above.
});
```

Подпишитесь и дождитесь данных перед отображением шаблона

```
Router.route('/waiting-first', {
    waitOn: function() {
        //subscribes to a publication
        //shows loading template until subscription is ready
        return Meteor.subscribe('somePublication')
    },

    action: function() {
        //render like above examples
    }
});
```

Подпишитесь на несколько публикаций и дождитесь получения данных до создания шаблона

```
Router.route('/waiting-first', {
    waitOn: function() {
        //subscribes to a publication
        //shows loading template until subscription is ready
        return [Meteor.subscribe('somePublication1'),Meteor.subscribe('somePublication2')];
    },

    action: function() {
        //render like above examples
    }
});
```

Руководство для железного маршрутизатора: http://iron-meteor.github.io/iron-router/

С FlowRouter

FlowRouter более модульный по сравнению с Iron Router.

Установить FlowRouter

```
meteor add kadira:flow-router
```

Оформление шаблона

В частности, вы должны вручную добавить пакет рендеринга макета для связи с вашим

механизмом рендеринга:

- Blaze Layout для Blaze: `meteor add kadira:blaze-layout`
- React Layout for React: `meteor add kadira:react-layout`

Затем вы можете визуализировать динамические шаблоны (в случае Blaze):

```
<template name="mainLayout">
  {{> Template.dynamic template=area}}
</template>
```

```
FlowRouter.route('/blog/:postId', {
  action: function (params) {
    BlazeLayout.render("mainLayout", {
      area: "blog"
    });
  }
});
```

Предоставление шаблона с параметрами и / или запросом

Параметры указаны на маршруте, например, с помощью Iron Router:

```
FlowRouter.route("/blog/:catId/:postId", {
  name: "blogPostRoute",
  action: function (params) {
    //...
  }
})
```

Но параметры не передаются в качестве контекста данных в дочерний шаблон. Вместо этого дочерний шаблон должен их прочитать:

```
// url: /blog/travel/france?showcomments=yes
var catId = FlowRouter.getParam("catId"); // returns "travel"
var postId = FlowRouter.getParam("postId"); // returns "france"

var color = FlowRouter.getQueryParam("showcomments"); // returns "yes"
```

Прочитайте маршрутизация онлайн: https://riptutorial.com/ru/meteor/topic/5119/маршрутизация

глава 23: Метеор + Реакция

замечания

React - это библиотека JavaScript для создания пользовательских интерфейсов. Это с открытым исходным кодом , разработанный и поддерживаемый Facebook. «Метеор» имеет готовую поддержку React.

Ресурсы:

- Реагировать на учебник
- Учебник Meteor + React

Examples

Setup и "Hello World"

Добавить Реагировать на ваш проект:

```
meteor npm install --save react react-dom react-mounter
```

Создайте файл `client/helloworld.jsx` чтобы отобразить простой компонент React:

```
import React, { Component } from 'react';
import { mount } from 'react-mounter';

// This component only renders a paragraph containing "Hello World!"
class HelloWorld extends Component {
  render() {
    return <p>Hello World!</p>;
  }
}

// When the client application starts, display the component by mounting it to the DOM.
Meteor.startup(() => {
  mount(HelloWorld);
});
```

Создайте реактивный контейнер, используя createContainer

Предположим, что есть коллекция под названием `Todos` и `autopublish` пакет `autopublish` . Вот основной компонент.

```
import { createContainer } from 'meteor/react-meteor-data';
import React, { Component, PropTypes } from 'react';
import Todos from '/imports/collections/Todos';
```

```
export class List extends Component {
  render() {
    const { data } = this.props;
    return (
      <ul className="list">
        {data.map(entry => <li {...entry} />)}
      </ul>
    )
  }
}

List.propTypes = {
  data: PropTypes.array.isRequired
};
```

Внизу вы можете добавить контейнер для подачи реактивных данных в компонент. Это будет выглядеть так.

```
export default createContainer(() => {
  return {
    data: Todos.find().fetch()
  };
}, List);
```

Отображение коллекции MongoDB

В этом примере показано, как коллекция MongoDB может отображаться в компоненте React. Коллекция постоянно синхронизируется между сервером и клиентом, и страница мгновенно обновляется по мере изменения содержимого базы данных.

Чтобы подключить компоненты React и коллекции Meteor, вам понадобится пакет `react-meteor-data` .

```
$ meteor add react-meteor-data
$ meteor npm install react-addons-pure-render-mixin
```

Простая коллекция объявляется `both/collections.js` . Каждый исходный файл в `both` каталогах является как клиентским, так и серверным кодом:

```
import { Mongo } from 'meteor/mongo';

// This collection will contain a list of random numbers
export const Numbers = new Mongo.Collection("numbers");
```

Сбор должен быть опубликован на сервере. Создайте простую публикацию на `server/publications.js` :

```
import { Meteor } from 'meteor/meteor';
import { Numbers } from '/both/collections.js';

// This publication synchronizes the entire 'numbers' collection with every subscriber
Meteor.publish("numbers/all", function() {
```

```
  return Numbers.find();
});
```

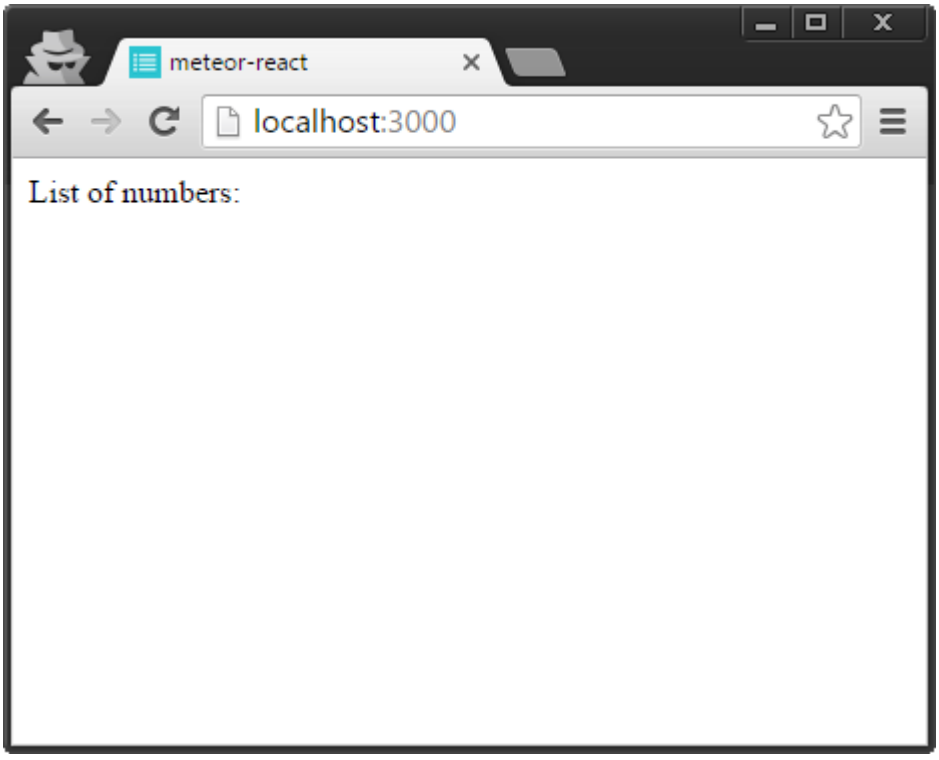
Используя функцию `createComponent` мы можем передать реактивные значения (например, коллекцию `Numbers`) в компонент React. `client/shownumbers.jsx` :

```
import React from 'react';
import { createContainer } from 'meteor/react-meteor-data';
import { Numbers } from '/both/collections.js';

// This stateless React component renders its 'numbers' props as a list
function _ShowNumbers({numbers}) {
  return <div>List of numbers:
    <ul>
      // note, that every react element created in this mapping requires
      // a unique key - we're using the _id auto-generated by mongodb here
      {numbers.map(x => <li key={x._id}>{x.number}</li>)}
    </ul>
  </div>;
}

// Creates the 'ShowNumbers' React component. Subscribes to 'numbers/all' publication,
// and passes the contents of 'Numbers' as a React property.
export const ShowNumbers = createContainer(() => {
  Meteor.subscribe('numbers/all');
  return {
    numbers: Numbers.find().fetch(),
  };
}, _ShowNumbers);
```

Первоначально база данных, вероятно, пуста.

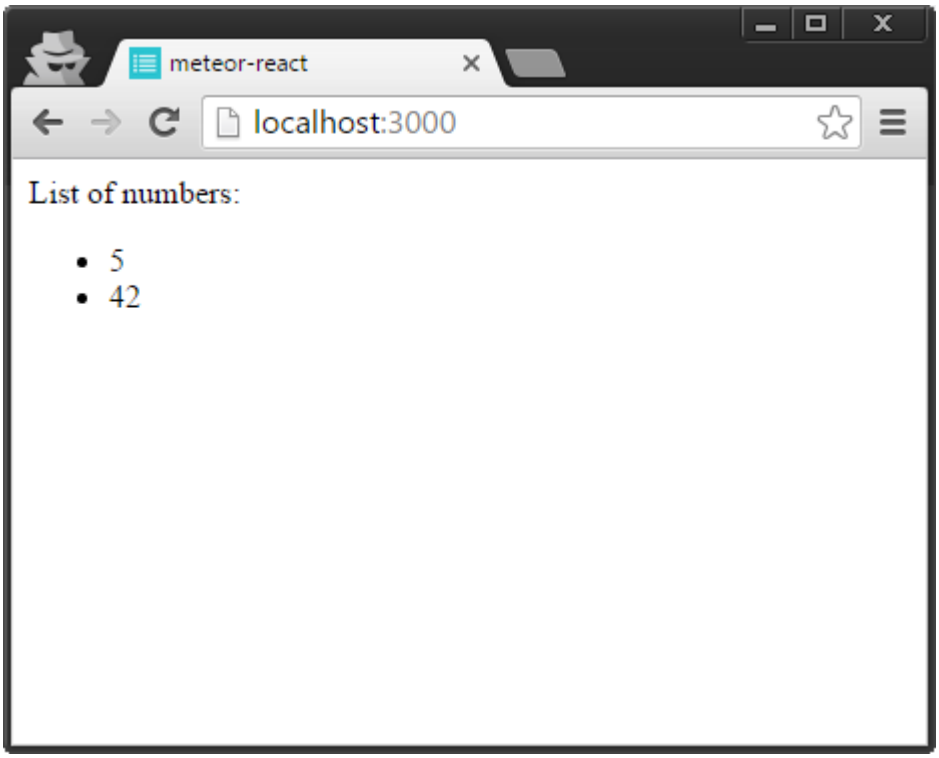


Добавьте записи в MongoDB и следите за обновлениями страницы автоматически.

```
$ meteor mongo
MongoDB shell version: 3.2.6
connecting to: 127.0.0.1:3001/meteor

meteor:PRIMARY> db.numbers.insert({number: 5});
WriteResult({ "nInserted" : 1 })

meteor:PRIMARY> db.numbers.insert({number: 42});
WriteResult({ "nInserted" : 1 })
```



Прочитайте Метеор + Реакция онлайн: https://riptutorial.com/ru/meteor/topic/3121/метеор-plus-реакция

глава 24: Миграция Монго Схемы

замечания

Часто бывает необходимо запустить сценарии обслуживания в вашей базе данных. Поля переименовываются; изменения структуры данных; функции, которые вы использовали для поддержки, удаляются; услуги мигрируют. Список причин, по которым вы, возможно, захотите изменить свою схему, довольно безграничен. Итак, «почему» довольно понятно.

«Как» немного незнакомец. Для тех людей, которые привыкли к функциям SQL, приведенные выше сценарии базы данных будут выглядеть странно. Но обратите внимание, как все они находятся в javascript, и как они используют тот же API, что и в Meteor, как на сервере, так и на клиенте. У нас есть совместимый API через нашу базу данных, сервер и клиент.

Запустите команды миграции схемы из оболочки метеорита mongo:

```
# run meteor
meteor

# access the database shell in a second terminal window
meteor mongo
```

Examples

Добавить поле версии для всех записей в коллекции

```
db.posts.find().forEach(function(doc){
    db.posts.update({_id: doc._id}, {$set:{'version':'v1.0'}}, false, true);
});
```

Удалить массив из всех записей в коллекции

```
db.posts.find().forEach(function(doc){
    if(doc.arrayOfObjects){
        // the false, true at the end refers to $upsert, and $multi, respectively
        db.accounts.update({_id: doc._id}, {$unset: {'arrayOfObjects': "" }}, false, true);
    }
});
```

Переименовать коллекцию

```
db.originalName.renameCollection("newName" );
```

Найти поле, содержащее определенную строку

С силой regex приходит большая ответственность ....

```
db.posts.find({'text': /.*foo.*|.*bar.*/i})
```

Создать новое поле из старого

```
db.posts.find().forEach(function(doc){
    if(doc.oldField){
        db.posts.update({_id: doc._id}, {$set:{'newField':doc.oldField}}, false, true);
    }
});
```

Вытаскивать объекты из массива и места в новом поле

```
db.posts.find().forEach(function(doc){
    if(doc.commenters){
        var firstCommenter = db.users.findOne({'_id': doc.commenters[0]._id });
        db.clients.update({_id: doc._id}, {$set:{'firstPost': firstCommenter }}, false, true);

        var firstCommenter = db.users.findOne({'_id': doc.commenters[doc.commenters.length -
1]._id });
        db.clients.update({_id: doc._id}, {$set:{'lastPost': object._id }}, false, true);
    }
});
```

Запись Blob из одной коллекции в другую коллекцию (т. E. Remove Join & Flatten)

```
db.posts.find().forEach(function(doc){
    if(doc.commentsBlobId){
        var commentsBlob = db.comments.findOne({'_id': commentsBlobId });
        db.posts.update({_id: doc._id}, {$set:{'comments': commentsBlob }}, false, true);
    }
});
```

Сделать конечно поле

```
db.posts.find().forEach(function(doc){
    if(!doc.foo){
        db.posts.update({_id: doc._id}, {$set:{'foo':''}}, false, true);
    }
});
```

Сделать поле «Конечно» имеет конкретную ценность

```
db.posts.find().forEach(function(doc){
    if(!doc.foo){
```

```
        db.posts.update({_id: doc._id}, {$set:{'foo':'bar'}}, false, true);
    }
});
```

Удалить запись, если конкретное поле является конкретным значением

```
db.posts.find().forEach(function(doc){
    if(doc.foo === 'bar'){
        db.posts.remove({_id: doc._id});
    }
});
```

Изменение определенного значения поля для нового значения

```
db.posts.find().forEach(function(doc){
    if(doc.foo === 'bar'){
        db.posts.update({_id: doc._id}, {$set:{'foo':'squee'}}, false, true);
    }
});
```

Отменить определенное поле до нулевого

```
db.posts.find().forEach(function(doc){
    if(doc.oldfield){
        // the false, true at the end refers to $upsert, and $multi, respectively
        db.accounts.update({_id: doc._id}, {$unset: {'oldfield': "" }}, false, true);
    }
});
```

Преобразовать ObjectId в String

```
db.posts.find().forEach(function(doc){
     db.accounts.update({_id: doc._id}, {$set: {'_id': doc._id.str }}, false, true);
});
```

Преобразование значений полей из чисел в строки

```
var newvalue = "";
db.posts.find().forEach(function(doc){
     if(doc.foo){
         newvalue = '"' + doc.foo + '"';
         db.accounts.update({_id: doc._id}, {$set: {'doc.foo': newvalue}});
     }
});
```

Преобразование значений полей из строк в числа

```
var newvalue = null;
```

```
db.posts.find().forEach(function(doc){
     if(doc.foo){
         newvalue = '"' + doc.foo + '"';
         db.accounts.update({_id: doc._id}, {$set: {'doc.foo': newvalue}});
     }
});
```

Создайте временную метку из ObjectID в поле _id

```
db.posts.find().forEach(function(doc){
    if(doc._id){
        db.posts.update({_id: doc._id}, {$set:{ timestamp: new
Date(parseInt(doc._id.str.slice(0,8), 16) *1000) }}, false, true);
    }
});
```

Создайте ObjectID из объекта Date

```
var timestamp = Math.floor(new Date(1974, 6, 25).getTime() / 1000);
var hex       = ('00000000' + timestamp.toString(16)).substr(-8); // zero padding
var objectId  = new ObjectId(hex + new ObjectId().str.substring(8));
```

Найти все записи, содержащие элементы в массиве

То, что мы здесь делаем, ссылается на индекс массива с использованием точечной нотации

```
db.posts.find({"tags.0": {$exists: true }})
```

Прочитайте Миграция Монго Схемы онлайн: https://riptutorial.com/ru/meteor/topic/3708/миграция-монго-схемы

глава 25: Мобильные приложения

Examples

Макет страницы на разных устройствах - CSS

Если ваше приложение будет запускаться на разных устройствах, оно должно будет отображаться на разных портах ViewPorts на основе размера устройства. Вы можете справиться с этим двумя способами: с правилами javascript или стилями мультимедиа CSS. Если вы использовали библиотеку MVC или MVVM, такую как Angular или Ember (или Blaze, если на то пошло), и только таргетинг на одну аппаратную или аппаратную платформу, вам может потребоваться переосмыслить вашу модель MVC как другое оборудование. ViewPorts введенные в ваше приложение.

```
// desktop
@media only screen and (min-width: 960px) {
}

// landscape orientation
@media only screen and (min-width: 768px) {
}

// portrait orientation
@media only screen and (min-width: 480px) {
}
```

Вам нужно выяснить, хотите ли вы стирать стили в 768px (портретный режим) или 1024 пикселя (пейзаж). Это предполагает, что вашим целевым мобильным устройством является iPad, который использует соотношение 3: 4. В противном случае вам нужно будет определить пропорции устройств, на которые вы хотите настроить таргетинг, и определить уровни порога оттуда.

Исправленные окна

Если вы планируете создавать макеты с экранами фиксированного размера для разных мобильных устройств, вы можете захотеть отразить этот дизайн при запуске приложения на рабочем столе. Следующий метод фиксирует размер окна OUTSIDE of PhoneGap, предоставляя окно на фиксированном уровне на рабочем столе. Иногда проще управлять ожиданиями пользователей и дизайном пользовательского интерфейса, ограничивая возможности!

```
// create a window of a specific size
var w=window.open('','', 'width=100,height=100');
w.resizeTo(500,500);

// prevent window resize
```

```
var size = [window.width,window.height];  //public variable
$(window).resize(function(){
    window.resizeTo(size[0],size[1]);
});
```

Автономное кэширование

Чтобы получить все это для работы, вам, вероятно, понадобится автономная поддержка, что означает кэширование данных приложений и пользовательских данных.

```
meteor add appcache
meteor add grounddb
```

Отключить прокрутку

В настольных приложениях вы можете отключить прокрутку, чтобы дать вашему приложению более привычное ощущение. Вы можете сделать это с помощью javascript, отключив, как браузер управляет DOM:

```
// prevent scrolling on the whole page
// this is not meteorish; TODO: translate to meteor-centric code
document.ontouchmove = function(e) {e.preventDefault()};

// prevent scrolling on specific elements
// this is not meteorish; TODO: translate to meteor-centric code
scrollableDiv.ontouchmove = function(e) {e.stopPropagation()};
```

Кроме того, вы можете использовать CSS, стили переполнения и прокрутки.

```
#appBody {
  overflow: hidden;
}

#contentContainer {
  .content-scrollable {
    overflow-y: auto;
    -webkit-overflow-scrolling: touch;
  }
}
```

Объектная модель, необходимая для работы выше, выглядит примерно так:

```
<div id="appBody">
  <div id="contentContainer">
    <div class="content-scrollable">
      <!-- content -->
    </div>
  </div>
</div>
```

Мультитач и жесты

На мобильных устройствах обычно нет клавиатур, поэтому вам нужно добавить в ваше приложение некоторые тактильные контроллеры. Два популярных пакета, которые, по-видимому, используют люди, - FastClick и Hammer. Установка проста.

```
meteor add fastclick
meteor add hammer:hammer
```

FastClick практически не нуждается в настройке, в то время как Hammer требует немного работы для подключения. Кононический пример приложения Todos выглядит следующим образом:

```
Template.appBody.onRendered(function() {
  if (Meteor.isCordova) {
    // set up a swipe left / right handler
    this.hammer = new Hammer(this.find('#appBody'));
    this.hammer.on('swipeleft swiperight', function(event) {
      if (event.gesture.direction === 'right') {
        Session.set(MENU_KEY, true);
      } else if (event.gesture.direction === 'left') {
        Session.set(MENU_KEY, false);
      }
    });
  }
});
```

Создайте свои иконки и заставки экрана

Прежде чем компилировать приложение и запустить его на своем устройстве, вам нужно создать несколько значков и заставки и добавить файл `mobile-config.js` в ваше приложение.

```
App.icons({
  // iOS
  'iphone': 'resources/icons/icon-60x60.png',
  'iphone_2x': 'resources/icons/icon-60x60@2x.png',
  'ipad': 'resources/icons/icon-72x72.png',
  'ipad_2x': 'resources/icons/icon-72x72@2x.png',

  // Android
  'android_ldpi': 'resources/icons/icon-36x36.png',
  'android_mdpi': 'resources/icons/icon-48x48.png',
  'android_hdpi': 'resources/icons/icon-72x72.png',
  'android_xhdpi': 'resources/icons/icon-96x96.png'
});

App.launchScreens({
  // iOS
  'iphone': 'resources/splash/splash-320x480.png',
  'iphone_2x': 'resources/splash/splash-320x480@2x.png',
  'iphone5': 'resources/splash/splash-320x568@2x.png',
  'ipad_portrait': 'resources/splash/splash-768x1024.png',
  'ipad_portrait_2x': 'resources/splash/splash-768x1024@2x.png',
  'ipad_landscape': 'resources/splash/splash-1024x768.png',
  'ipad_landscape_2x': 'resources/splash/splash-1024x768@2x.png',
```

```
  // Android
  'android_ldpi_portrait': 'resources/splash/splash-200x320.png',
  'android_ldpi_landscape': 'resources/splash/splash-320x200.png',
  'android_mdpi_portrait': 'resources/splash/splash-320x480.png',
  'android_mdpi_landscape': 'resources/splash/splash-480x320.png',
  'android_hdpi_portrait': 'resources/splash/splash-480x800.png',
  'android_hdpi_landscape': 'resources/splash/splash-800x480.png',
  'android_xhdpi_portrait': 'resources/splash/splash-720x1280.png',
  'android_xhdpi_landscape': 'resources/splash/splash-1280x720.png'
});
```

Трубопровод архитектуры Метеор Кордова

Теперь пришло время пройти через документацию интеграции Metega Cordova Phonegap Integration .

Поскольку эта документация была написана, были выпущены XCode и Yosemite, что вызвало некоторые икоты при установке. Вот шаги, которые мы должны были выполнить, чтобы получить Meteor, скомпилированный для устройства iOS.

- Поднимитесь до Йосемити.
- Удалить XCode (перетащить из папки «Приложения» в корзину)
- Установите XCode 6.1 из магазина приложений.
- Соглашайтесь на различные условия.

```
# 5.  clone and rebuild the ios-sim locally
#     (this step will not be needed in future releases)
git clone https://github.com/phonegap/ios-sim.git
cd ios-sim
rake build

# 6.  make sure we can update the .meteor/packages locations
#     (this step will not be needed in future releases)
sudo chmod -R 777 ~/.meteor/packages

# 7.  copy the new build into Meteor locations
#     (this step will not be needed in future releases)
for i in `find ~/.meteor/packages/meteor-tool/ -name ios-sim -type f`; do
  cp -R ./build/Release/ios-sim "$i"
done

# 8.  install the ios platform to your app
cd myapp
meteor list-platforms
meteor add-platform ios
meteor list-platforms

# 9.  and that there aren't dead processes
ps -ax
kill -9 <pid>
# /Users/abigailwatson/.meteor/packages/meteor-
tool/.1.0.35.wql4jh++os.osx.x86_64+web.browser+web.cordova/meteor-tool-
os.osx.x86_64/dev_bundle/mongodb/bin/mongod
# tail -f /Users/abigailwatson/Code/Medstar/dart/webapp/.meteor/local/cordova-
```

```
build/platforms/ios/cordova/console.log

# 10.  make sure there are correct permissions on the application (important!)
sudo chmod -R 777 .meteor/local/

# 11.  run app
meteor run ios

# 12.  if that doesn't work, clear the directory
sudo rm -rf .meteor/local

# 13a.  run meteor again to create the default browser build
meteor

# 13b.  run it a second time so bootstrap and other packages get downloaded into the browser build
ctrl-x
meteor

# 14.  then run the ios version
ctrl-x
meteor run ios
```

XCode должен запускаться во время процесса. Выберите ваш симулятор и нажмите кнопку «Воспроизвести».

Разработка IOS

- Зарегистрируйте свою учетную запись Apple Developer
- Зарегистрируйте идентификатор приложения для своего приложения.
- Зарегистрируйте UUID ваших тестовых устройств
- Создайте профиль обеспечения разработки приложений iOS
    - Создание сертификатаSigningRequest из KeychainAccess
    - Отправить CertificateSigningRequest в https://developer.apple.com/account/ios/profile/profileCreate.action
    - Загрузите и дважды щелкните сертификат для импорта в Keychain
- Перейдите в раздел XCode> Настройки> Аккаунты и зарегистрируйте свою учетную запись Apple Developer

Тестирование устройств IOS

- Убедитесь, что ваша рабочая станция разработки и iPhone подключены к той же сети WiFi. Модем, горячие точки и другие специальные сети не будут работать.
- Запустить `sudo meteor run ios-device` запуска `sudo meteor run ios-device`
- Разверните на свое устройство!

Настройте проект Cordova (config.xml)

Meteor считывает файл `mobile-config.js` в корневой каталог вашего приложения во время сборки и использует настройки, указанные там, для создания `config.xml` Кордовы.

```
Project_folder
 ├── /.meteor
 └── mobile-config.js
```

Большинство конфигураций могут быть достигнуты с помощью `mobile-config.js` (метаданные приложений, предпочтений, значков и экранов запуска, а также параметры установки плагинов Кордовы).

```
App.info({
  id: 'com.example.matt.uber',
  name: 'über',
  description: 'Get über power in one button click',
  author: 'Matt Development Group',
  email: 'contact@example.com',
  website: 'http://example.com'
});

// Set up resources such as icons and launch screens.
App.icons({
  'iphone': 'icons/icon-60.png',
  'iphone_2x': 'icons/icon-60@2x.png',
  // ... more screen sizes and platforms ...
});

App.launchScreens({
  'iphone': 'splash/Default~iphone.png',
  'iphone_2x': 'splash/Default@2x~iphone.png',
  // ... more screen sizes and platforms ...
});

// Set PhoneGap/Cordova preferences
App.setPreference('BackgroundColor', '0xff0000ff');
App.setPreference('HideKeyboardFormAccessoryBar', true);
App.setPreference('Orientation', 'default');
App.setPreference('Orientation', 'all', 'ios');

// Pass preferences for a particular PhoneGap/Cordova plugin
App.configurePlugin('com.phonegap.plugins.facebookconnect', {
  APP_ID: '1234567890',
  API_KEY: 'supersecretapikey'
});
```

Не вручную отредактируйте файл `/.meteor/local/cordova-build/config.xml` , так как он будет регенерирован при каждом `meteor run ios/android` или `meteor build` , выполняемом `meteor build` , поэтому вы потеряете все свои модификации.

Справочная страница: Руководство Meteor> Build> Mobile> Настройка вашего приложения

Обнаружение события deviceready

Разумеется, лучший способ обнаружить мобильный телефон - это то, что оборудование должно уведомить вас напрямую. Cordova PhoneGap предоставляет событие « deviceready», к которому вы можете добавить прослушиватель событий.

```
document.addEventListener('deviceready', function(){
 Session.set('deviceready', true);
}, false);
```

Прочитайте Мобильные приложения онлайн: https://riptutorial.com/ru/meteor/topic/3705/мобильные-приложения

глава 26: Модули ES2015 (импорт и экспорт)

замечания

Документация MDN для импорта: https://developer.mozilla.org/en/docs/web/javascript/reference/statements/import Документация MDN для экспорта: https://developer.mozilla.org/en/docs/web/javascript / reference / statements / export ExploringJS глава по модулям: http://exploringjs.com/es6/ch_modules.html

Examples

Импорт в модули приложений

Узел модулей

```
import url from 'url';
import moment from 'moment';
```

Метеоритные пакеты

```
import { Meteor } from 'meteor/meteor';
import { SimpleSchema } from 'meteor/aldeed:simple-schema';
```

Импорт в пакеты Meteor

В package.js:

```
Npm.depends({
  moment: "2.8.3"
});
```

В файле пакета:

```
import moment from 'moment';
```

Экспорт переменных из модулей приложения

```
// Default export
export default {};

// Named export
export const SomeVariable = {};
```

Экспорт символов из пакетов Meteor

В файле mainModule :

```
export const SomeVar = {};
```

Прочитайте Модули ES2015 (импорт и экспорт) онлайн:
https://riptutorial.com/ru/meteor/topic/3763/модули-es2015--импорт-и-экспорт-

глава 27: Монгольские коллекции

замечания

Полезный способ подумать о коллекциях Монго в терминах Who, What, When, Where, Why и How. У Mongo есть следующие оптимизации для разных типов данных:

Где - GeoJSON
Когда - временные метки ObjectID
Кто - строки счетчика метеоров
How - JSON для деревьев решений

Который оставляет документ по умолчанию в Монго, грубо говоря, «Что».

Examples

Создание записей в устаревшей базе данных

Вы можете по умолчанию использовать обычный формат Mongo, указав свои коллекции в поле idGeneration.

```
MyCollection = new Meteor.Collection('mycollection', {idGeneration : 'MONGO'});
```

Вставка данных в документ

Многие новички в Mongo борются с основами, например, как вставить массив, дату, логическое значение, переменную сеанса и т. Д. В запись документа. В этом примере приведены некоторые рекомендации по основным входам данных.

```
Todos.insert({
  text: "foo",                        // String
  listId: Session.get('list_id'),     // String
  value: parseInt(2),                 // Number
  done: false,                        // Boolean
  createdAt: new Date(),              // Dimestamp
  timestamp: (new Date()).getTime(),  // Time
  tags: []                            // Array
});
```

Получение _id из недавно созданного документа

Вы можете получить его либо синхронно:

```
var docId = Todos.insert({text: 'foo'});
console.log(docId);
```

Или асинхронно:

```
Todos.insert({text: 'foo'}, function(error, docId){
  console.log(docId);
});
```

Временные данные

Использование MongoDB для данных временных рядов - очень хороший документ и установленный прецедент с официальными документами и презентациями. Прочтите и посмотрите официальную документацию от MongoDB, прежде чем пытаться изобрести свои собственные схемы для данных временных рядов.

MongoDB для данных временных рядов

В общем, вы захотите создать «ведра» для ваших данных таймера:

```
DailyStats.insert({
   "date" : moment().format("MM-DD-YYYY"),
   "dateIncrement" : moment().format("YYYYMMDD"),
   "dailyTotal" : 0,
   'bucketA': 0,
   'bucketB': 0,
   'bucketC': 0
   });
```

Затем добавьте эти ведра в качестве фидов данных в ваше приложение. Этот прирост можно поместить в методе Метеор, наблюдателе коллекции, конечной точке REST API и в других местах.

```
DailyStats.update({_id: doc._id}, {$inc: {bucketA: 1} });
```

Более подробный пример «Метеор» см. В примерах из трека «Clinical Meteor»:

Конвейер аналитики Realtime Analytics
Клинические метеориты - графики - Дайлистаты

Фильтрация с помощью регулярных выражений

Простой шаблон для фильтрации подписки на сервере, с использованием регулярных выражений, реактивных переменных сеанса и отладки автозапуска.

```
// create our collection
WordList =  new Meteor.Collection("wordlist");

// and a default session variable to hold the value we're searching for
Session.setDefault('dictionary_search', '');

Meteor.isClient(function(){
```

```
    // we create a reactive context that will rerun items when a Session variable gets updated

    Deps.autorun(function(){
        // and create a subscription that will get re-subscribe to when Session variable gets
updated
        Meteor.subscribe('wordlist', Session.get('dictionary_search'));
    });

    Template.dictionaryIndexTemplate.events({
        'keyup #dictionarySearchInput': function(evt,tmpl){
            // we set the Session variable with the value of our input when it changes
            Session.set('dictionary_search', $('#dictionarySearchInput').val());
        },
        'click #dictionarySearchInput':function(){
            // and clear the session variable when we enter the input
            Session.set('dictionary_search', '');
        },
    });
});
Meteor.isServer(function(){
    Meteor.publish('wordlist', function (word_search) {
        // this query gets rerun whenever the client subscribes to this publication
        return WordList.find({
            // and here we do our regex search
            Word: { $regex: word_search, $options: 'i' }
        },{limit: 100});
    });
});
```

И HTML, который используется на клиенте:

```
<input id="dictionarySearchInput" type="text" placeholder="Filter..." value="hello"></input>
```

Этот шаблон сам по себе довольно прямолинейный, но регулярных выражений может и не быть. Если вы не знакомы с регулярными выражениями, вот несколько полезных руководств и ссылок:

Учебное пособие по регулярному выражению
Шрифт регулярного выражения
Регулярные выражения в Javascript

Геопространственные коллекции - больше обучения

Геопространственные коллекции обычно включают хранение GeoJSON в базе данных Mongo, передачу этих данных клиенту, доступ к `window.navigator.geolocation` , загрузку API Карт, преобразование GeoJSON в LatLngs и построение графика на карте. Предпочтительно все в режиме реального времени. Вот список ресурсов, которые помогут вам начать работу:

- mongodb оптимально сохраняет свои данные в geoJSON
- geojson.org
- window.navigator.geolocation

- Геолокация HTML
- API карт API
- google.maps.LatLng
- Google map.data.loadGeoJson
- Метеор-Cordova-геолокации-фон
- PhoneGap-GoogleMaps-плагин
- LatLng
- maps.documentation
- google.maps.LatLng
- Индексы 2dsphere
- создать индекс 2dsphere
- запросить индекс 2dsphere
- геопространственные индексы и запросы

Запросы аудиторской коллекции

В следующем примере будут записываться все запросы на сборку на консоль сервера в реальном времени.

```
Meteor.startup(
  function () {
    var wrappedFind = Meteor.Collection.prototype.find;

    // console.log('[startup] wrapping Collection.find')

    Meteor.Collection.prototype.find = function () {
      // console.log(this._name + '.find', JSON.stringify(arguments))
      return wrappedFind.apply(this, arguments);
    }
  },

  function () {
    var wrappedUpdate = Meteor.Collection.prototype.update;

    // console.log('[startup] wrapping Collection.find')

    Meteor.Collection.prototype.update = function () {
      console.log(this._name + '.update', JSON.stringify(arguments))
      return wrappedUpdate.apply(this, arguments);
    }
  }
);
```

Наблюдатели и рабочие функции

Если цикл события Node действует как цепь велосипеда, наблюдатель на стороне сервера похож на переключатель. Это механизм передачи, который будет собираться в сбор данных по мере поступления данных. Он может быть очень эффективным, так как у всех гоночных велосипедов есть переключатели. Но это также источник для разрушения всей системы. Это высокоскоростная реактивная функция, которая может взорвать вас. Имейте

в виду.

```
Meteor.startup(function(){
  console.log('starting worker....');

  var dataCursor = Posts.find({viewsCount: {$exists: true}},{limit:20});

  var handle = dataCursor.observeChanges({
    added: function (id, record) {
      if(record.viewsCount > 10){
         // run some statistics
         calculateStatistics();

         // or update a value
         Posts.update({_id: id}, {$set:{
           popular: true
         }});

      }
    },
    removed: function () {
      console.log("Lost one.");
    }
  });
});
```

Обратите внимание, что предел 20 - это размер переключателя ... сколько зубов у него; или, точнее, сколько предметов находится в курсоре, поскольку оно идет по коллекции. Будьте осторожны с использованием ключевого слова «var» в этом виде функции. Запишите как можно меньше объектов в память и сосредоточьтесь на повторном использовании объектов внутри добавленного метода. Когда opslog включен, и эта вещь идет полным ходом, это главный кандидат для выявления неприятных утечек памяти, если он записывает объекты в кучу памяти быстрее, чем сборщик мусора узла способен очищать вещи.

Вышеупомянутое решение не будет масштабироваться горизонтально, потому что каждый экземпляр Meteor будет пытаться обновить одну и ту же запись. Таким образом, для такого масштабирования горизонтально требуется какое-то определение окружающей среды.

См. Пакет `percolatestudios:synced-cron` для отличного примера синхронизации рабочих сервисов на нескольких компьютерах в кластере.
Метеор-синхронизируются-хрон

Прочитайте Монгольские коллекции онлайн: https://riptutorial.com/ru/meteor/topic/5120/монгольские-коллекции

глава 28: Наборы реплик и осколки

замечания

Для тех, кто не знаком, набор реплик определяется как избыточная конфигурация трех серверов. Заштрихованная база данных определяется как горизонтально взломанная база данных, где каждый осколок определяется как набор реплик. Поэтому кластер Mongo с кластером включает в себя как минимум 11 серверов для кластера с двумя осколками и увеличивается на три сервера для каждого дополнительного осколка. Таким образом, у осколочного кластера всегда есть экземпляры сервера 11, 14, 17, 20, 23 и т. Д. То есть, есть 2 осколка по 3 сервера каждый, еще 3 конфигурационных контроллера и 2 маршрутизатора. Всего 11 серверов для кластера с двумя кланами.

Examples

Набор копий Quickstart

Создайте себе **три** сервера, используя любое физическое или виртуальное оборудование, которое вы пожелаете. (В этом руководстве предполагается, что вы используете Ubuntu в качестве вашей операционной системы.) Затем повторите следующие инструкции три раза ... один раз для каждого сервера.

```
# add the names of each server to the host file of each server
sudo nano /etc/hosts
  10.123.10.101 mongo-a
  10.123.10.102 mongo-b
  10.123.10.103 mongo-c

# install mongodb on the server
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv 7F0CEB10
echo 'deb http://downloads-distro.mongodb.org/repo/ubuntu-upstart dist 10gen' | sudo tee
/etc/apt/sources.list.d/mongodb.list
sudo apt-get update
sudo apt-get install mongodb-10gen


# create the /data/ directories
sudo mkdir /data
sudo mkdir /data/logs
sudo mkdir /data/db

# make sure the mongodb user and group have access to our custom directories
sudo chown -R mongodb:mongodb /data

# edit the mongo upstart file in /etc/init/mongodb.conf
sudo nano /etc/init/mongodb.conf
  start on started mountall
  stop on shutdown
  respawn
```

```
  respawn limit 99 5
  setuid mongodb
  setgid mongodb
  script
    exec /usr/bin/mongod --config /etc/mongodb.conf >> /data/logs/mongo-a.log 2>&1
  end script


# edit mongodb configuration file
sudo nano /etc/mongodb.conf
    dbpath=/data/db
    logpath=/data/logs/mongod.log
    logappend=true
    port=27017
    noauth=true
    replSet=meteor
    fork=true

# add a mongo log-rotation file
sudo nano /etc/logrotate.d/mongod
  /data/logs/*.log {
    daily
    rotate 30
    compress
    dateext
    missingok
    notifempty
    sharedscripts
    copytruncate
    postrotate
        /bin/kill -SIGUSR1 `cat /data/db/mongod.lock 2> /dev/null` 2> /dev/null || true
    endscript
  }

# make sure mongod service is started and running
sudo service mongodb start
sudo reboot
```

Конфигурация набора реплик

Затем перейдите в оболочку mongo и запустите набор реплик, например:

```
meteor mongo

  > rs.initiate()
  PRIMARY> rs.add("mongo-a")
  PRIMARY> rs.add("mongo-b")
  PRIMARY> rs.add("mongo-c")
  PRIMARY> rs.setReadPref('secondaryPreferred')
```

Прочитайте Наборы реплик и осколки онлайн: https://riptutorial.com/ru/meteor/topic/4332/наборы-реплик-и-осколки

глава 29: Настройка базового кода для автоматического тестирования

Examples

Коды установки

- Перейдите на сайт Codeship.com и создайте учетную запись (или войдите в систему)
- Создать новый проект
- Импортируйте свой проект через Github или Bitbucket
- На экране «Настроить ваши тесты» используйте следующие команды:
    - Выберите «Я хочу создать свои собственные команды» в раскрывающемся списке «Выберите свою технологию, чтобы предварительно заполнить основные команды».
    - Введите следующие команды:

```
curl -o meteor_install_script.sh https://install.meteor.com/
chmod +x meteor_install_script.sh
sed -i "s/type sudo >\/dev\/null 2>&1/\ false /g" meteor_install_script.sh
./meteor_install_script.sh
export PATH=$PATH:~/.meteor/
meteor --version
meteor npm install
```

    - Оставьте тестовые команды следующим образом:

```
npm test
```

- Нажмите новую фиксацию на Github / Bitbucket
- это оно

Подготовьте проект

- Напишите несколько тестов
- Установка отправки: мокко-фантомс :

```
meteor add dispatch:mocha-phantomjs
```

- Добавьте тестовую команду в пакет package.json.

```
{
  "name": "awesome meteor package",
  "scripts": {
```

```
    "test": "meteor test --driver-package dispatch:mocha-phantomjs --once"
  }
}
```

- Убедитесь, что вы можете запустить `npm test` в корневом каталоге проекта.

Прочитайте Настройка базового кода для автоматического тестирования онлайн: https://riptutorial.com/ru/meteor/topic/6741/настройка-базового-кода-для-автоматического-тестирования

глава 30: Настройка производительности

замечания

Следует отметить, что Meteor - это просто Javascript и Node.js. Да, это очень конкретная реализация этих двух технологий и имеет собственную уникальную экосистему, а также использует изоморфные API и хранилище данных JSON для достижения действительно потрясающих результатов. Но, в конце концов, Meteor - это веб-технология, и она написана в Javascript. Таким образом, все ваши типичные методы работы с javascript по-прежнему применяются. Начните там.

25 Javascript Performance Techniques
Оптимизация производительности для высокоскоростного Javascript
Оптимизация кода JavaScript
Рекомендации по производительности для JavaScript в V8
10 Javascript Performance Boosting Tip
Создать эффективный JavaScript
Улучшение эффективности ваших проектов JS Meteor

Examples

Проектирование и развертывание готового программного обеспечения

Помните, что все лучшие практики типичной веб-архитектуры все еще применяются. Для превосходного обзора по теме, пожалуйста, обратитесь к замечательной книге Майкла Нигара « Release It! Разработка и развертывание готового программного обеспечения . Написание вашего приложения в Meteor не освобождает вас от аудита сторонних библиотек, написания автоматических выключателей, обертывания вызовов в тайм-аутах, мониторинга ваших пулов ресурсов и всего остального. Если вы хотите, чтобы ваше приложение работало хорошо, вам нужно убедиться, что вы используете шаблоны стабильности и избегаете анти-шаблонов.

Шаблоны стабильности

- Таймауты
- Предохранители
- Перегородки
- Подтверждение связи
- Стабильность

Точки интеграции

- Сторонние библиотеки

- Масштабирующие эффекты
- Несбалансированные возможности
- Мощные анти-шаблоны

Конфликт пула ресурсов

- AJAX Overkill
- Просроченные сессии
- Чрезмерное белое пространство
- Учет данных

Если эти понятия незнакомы и не кажутся вам второстепенными, это означает, что вы не принесли больших производственных систем в Интернете. Купите копию книги. Это будет потраченное время и деньги.

Прочитайте Настройка производительности онлайн:
https://riptutorial.com/ru/meteor/topic/3363/настройка-производительности

глава 31: Непрерывная интеграция и облака устройств (с помощью Nightwatch)

замечания

Nightwatch предоставляет приемные и сквозные тесты для приложений Meteor с v0.5 дней и управляет миграциями от PHP до Spark to Blaze и React; и все основные платформы непрерывной интеграции. Дополнительную помощь можно найти в:

Документация API Nightwatch
Группа пользователей Nightwatch.js Google

Examples

Travis

Travis - это оригинальная служба непрерывной интеграции, которая стала популярной в сообществе Meteor. Он прочный и надежный, уже давно имеет хостинг-сервер с открытым исходным кодом и с годами запускает сотни тысяч тестов Nightwatch.

.travis.yml
Просто `.travis.yml` файл `.travis.yml` в корень вашего приложения, например:

```
# this travis.yml file is for the leaderboard-nightwatch example, when run standalone
language: node_js

node_js:
  - "0.10.38"

services:
  - mongodb

sudo: required

env:
  global:
    - TRAVIS=true
    - CONFIG_PREFIX=`npm config get prefix`
    - DISPLAY=:99.0
    - NODE_ENV=`travis`
  matrix:

cache:
  directories:
    - .meteor/local/build/programs/server/assets/packages
    - .meteor

before_install:
```

```
  # set up the node_modules dir, so we know where it is
  - "mkdir -p node_modules &"

  # install nightwatch, selenium, , so we can launch nightwatch and selenium
  - "meteor npm install nightwatch selenium-server-standalone-jar chromedriver"

  # fire up xvfb on port :99.0
  - "sh -e /etc/init.d/xvfb start"

  # set the xvfb screen size to 1280x1024x16
  - "/sbin/start-stop-daemon --start --quiet --pidfile /tmp/custom_xvfb_99.pid --make-pidfile
--background --exec /usr/bin/Xvfb -- :99 -ac -screen 0 1280x1024x16"

  # install meteor
  - "curl https://install.meteor.com | /bin/sh"

  # give meteor a few seconds after installing
  - "sleep 10"

  # setup Meteor app
  - "cd webapp"
  - "meteor &"

  # give Meteor some time to download packages, init data, and to start
  - "sleep 60"

# then run nightwatch using the chromedriver
script: "nightwatch -c .meteor/nightwatch.json"
```

Круг

Circle - это новейшая служба непрерывной интеграции, которая стала популярной среди метеоритов. У него есть все последние колокола и свистки, поскольку непрерывная интеграция идет. Следующий скрипт поддерживает множество новых функций, в том числе:

- скриншоты
- артефакты
- git подмодули
- обнаружение окружающей среды
- кеширование каталогов
- оптимизация параллелизма
- Скрипты npm
- непрерывное развертывание
- webhooks

.circle.yml

```
## Customize the test machine
machine:

  # Timezone
  timezone:
```

```
    America/Los_Angeles # Set the timezone

  # Add some environment variables
  environment:
    CIRCLE_ENV: test
    CXX: g++-4.8
    DISPLAY: :99.0
    NPM_PREFIX: /home/ubuntu/nvm/v0.10.33
    INITIALIZE: true
    NODE_ENV: circle


## Customize checkout
checkout:
 post:
   #- git submodule sync
   #- git submodule update --init --recursive # use submodules

general:
  build_dir: webapp
  artifacts:
    - "./tests/nightwatch/screenshots" # relative to the build directory

## Customize dependencies
dependencies:
  cache_directories:
    - "~/.meteor" # relative to the user's home directory
    - ~/nvm/v0.10.33/lib/node_modules/starrynight
    - ~/nvm/v0.10.33/bin/starrynight

  pre:
    # Install Starrynight unless it is cached
    - if [ ! -e ~/nvm/v0.10.33/bin/starrynight ]; then npm install -g starrynight; else echo
"Starrynight seems to be cached"; fi;
    # Install  Meteor
    - mkdir -p ${HOME}/.meteor
    # If Meteor is already cached, do not need to build it again.
    - if [ ! -e ${HOME}/.meteor/meteor ]; then curl https://install.meteor.com | /bin/sh; else
echo "Meteor seems to be cached"; fi;
    # Link the meteor executable into /usr/bin
    - sudo ln -s $HOME/.meteor/meteor /usr/bin/meteor
    # Check if the helloworld directory already exists, if it doesn't, create the helloworld
app
    # The following doesn't work, because it should be checking ${HOME}/active-
entry/helloworld
    # - if [ ! -e ${HOME}/helloworld ]; then meteor create --release METEOR@1.1.0.3
helloworld; else echo "helloworld app seems to be cached"; fi;

  override:
    #- meteor list

## Customize test commands
test:
  pre:
    #- starrynight fetch
    #- cd packages && rm -rf temp
    #- cd packages && ls -la
    #- starrynight autoconfig
    - meteor update --release METEOR@1.3.3
    - meteor npm install --save jquery bootstrap react react-dom react-router react-bootstrap
react-komposer
```

```
    - cat .meteor/nightwatch.json
    - meteor:
          background: true
    - sleep 60
  override:
    - meteor npm run-script nightwatch


## Customize deployment commands
#deployment:
#  production:
#    branch: master
#    commands:
#      - printf "<Meteor username>\n<Meteor password>\n" | meteor deploy myapp.meteor.com

## Custom notifications
#notify:
  #webhooks:
    # A list of hashes representing hooks. Only the url field is supported.
    #- url: https://someurl.com/hooks/circle
```

SauceLabs

SauceLabs - это автоматизированная тестовая платформа для предприятия. Он поддерживает как непрерывную интеграцию, кросс-браузерное тестирование, так и облако мобильных устройств. Затраты выше, чем у Travis, Circle или BrowserStack.

```
{
  "selenium" : {
    "start_process" : false,
    "host" : "ondemand.saucelabs.com",
    "port" : 80,
  },
  "test_settings" : {
    "chrome_saucelabs": {
      "selenium_host": "ondemand.saucelabs.com",
      "selenium_port": 80,
      "username": "${SAUCE_USERNAME}",
      "access_key": "${SAUCE_ACCESS_KEY}",
      "use_ssl": false,
      "silent": true,
      "output": true,
      "screenshots": {
        "enabled": false,
        "on_failure": true,
        "path": ""
      },
      "desiredCapabilities": {
        "name": "test-example",
        "browserName": "chrome"
      },
      "globals": {
        "myGlobal": "some_sauce_global"
      }
    },
  }
}
```

BrowserStack

BrowserStack использует облако устройств для кросс-браузерного тестирования. Цель состоит в том, чтобы разрешить тестирование сценариев Selenium на каждом устройстве.

```
{
  "selenium" : {
    "start_process" : false,
    "host" : "hub.browserstack.com",
    "port" : 80,
  },

  "test_settings" : {
    "default" : {
      "launch_url" : "http://hub.browserstack.com",
      "selenium_port"  : 80,
      "selenium_host"  : "hub.browserstack.com",
      "silent": true,
      "screenshots" : {
        "enabled" : false,
        "path" : "",
      },
      "desiredCapabilities": {
        "browserName": "firefox",
        "javascriptEnabled": true,
        "acceptSslCerts": true,
        "browserstack.user": "USERNAME",
        "browserstack.key": "KEY"
      }
    }
  }
}
```

Прочитайте Непрерывная интеграция и облака устройств (с помощью Nightwatch) онлайн: https://riptutorial.com/ru/meteor/topic/6550/непрерывная-интеграция-и-облака-устройств--с-помощью-nightwatch-

глава 32: Непрерывное развертывание в галактике из кодов

замечания

Эта тема в значительной степени вдохновлена Nate Strausers Миграция метеоритных приложений от модуля к галактике с непрерывным развертыванием из Кодекса .

Examples

Настроить

- Создайте `deployment_token.json` :

```
METEOR_SESSION_FILE=deployment_token.json meteor login
```

- Создайте следующие переменные среды в Codeship: (https://codeship.com/projects/PROJECT_NUMBER/configure_environment)
    - METEOR_TARGET: your.domain.com
    - METEOR_TOKEN: Копировать / Вставить содержимое deploy_token.json. Что-то вроде: `{"sessions": {"www.meteor.com": {"session": "12345 ...`
    - METEOR_SETTING: Скопируйте / Вставьте содержимое ваших настроек.json. Что-то вроде: `{"private": {...`
- Создайте новый конвейер развертывания здесь https://codeship.com/projects/YOUR_PROJECT_NUMBER/deployment_branches/new
    - Мы развертываем только ведущую ветвь. Итак, установите: Branch - это точно: master.
- Добавьте в качестве развертывания «Пользовательский сценарий» со следующим содержимым:

```
echo $METEOR_TOKEN > deployment_token.json
echo $METEOR_SETTINGS > deployment_settings.json
meteor npm prune --production
DEPLOY_HOSTNAME=galaxy.meteor.com METEOR_SESSION_FILE=deployment_token.json meteor deploy
$METEOR_TARGET --settings deployment_settings.json
```

Прочитайте Непрерывное развертывание в галактике из кодов онлайн: https://riptutorial.com/ru/meteor/topic/6743/непрерывное-развертывание-в-галактике-из-кодов

глава 33: Обертка асинхронных методов в волокно для синхронного выполнения.

Синтаксис

1. Meteor.wrapAsync (func, [context])

параметры

параметры	подробности
func: Функция	Асинхронная / синхронная функция, которая будет обернута в волокно, которое принимает обратный вызов w / parameters `(error, result)` .
контекст: Любой (необязательный)	Контекст данных, в котором функция выполняется.

замечания

Асинхронно завернутая функция все равно может выполняться асинхронно, если обратный вызов с параметрами `(error, result) => {}` задается как параметр для завернутой функции.

Включение `Meteor.wrapAsync` позволяет `Meteor.wrapAsync` код с обратными вызовами, учитывая, что обратными вызовами теперь можно пренебречь в компенсации за то, что блокировка звонка представляет собой существующее `Fiber` .

Чтобы понять, как работают Fibers, читайте здесь: https://www.npmjs.com/package/fibers .

Examples

Синхронное выполнение асинхронных методов NPM с обратными вызовами.

В этом примере обертывается асинхронный метод `oauth2.client.getToken(callback)` из пакета NPM пакета `simple-oauth2` в Fiber, чтобы метод можно было вызвать синхронно.

```
const oauth2 = require('simple-oauth2')(credentials);

const credentials = {
    clientID: '#####',
```

```
    clientSecret: '#####',
    site: "API Endpoint Here."
};

Meteor.startup(() => {
    let token = Meteor.wrapAsync(oauth2.client.getToken)({});
    if (token) {
        let headers = {
            'Content-Type': "application/json",
            'Authorization': `Bearer ${token.access_token}`
        }

        // Make use of requested OAuth2 Token Here (Meteor HTTP.get).
    }
});
```

Прочитайте Обертка асинхронных методов в волокно для синхронного выполнения. онлайн: https://riptutorial.com/ru/meteor/topic/2530/обертка-асинхронных-методов-в-волокно-для-синхронного-выполнения-

глава 34: Обнаружение окружающей среды

Examples

Расширенные настройки окружения

Для более сложных приложений вам нужно создать объект `` settings.json`, используя несколько переменных среды.

```
if(Meteor.isServer){
  Meteor.startup(function()){
    // this needs to be run on the server
    var environment, settings;

    environment = process.env.METEOR_ENV || "development";

    settings = {
      development: {
        public: {
          package: {
            name: "jquery-datatables",
            description: "Sort, page, and filter millions of records. Reactively.",
            owner: "LumaPictures",
            repo: "meteor-jquery-datatables"
          }
        },
        private: {}
      },
      staging: {
        public: {},
        private: {}
      },
      production: {
        public: {},
        private: {}
      }
    };

    if (!process.env.METEOR_SETTINGS) {
      console.log("No METEOR_SETTINGS passed in, using locally defined settings.");
      if (environment === "production") {
        Meteor.settings = settings.production;
      } else if (environment === "staging") {
        Meteor.settings = settings.staging;
      } else {
        Meteor.settings = settings.development;
      }
      console.log("Using [ " + environment + " ] Meteor.settings");
    }
  });
}
```

Определение параметров приложения с помощью METEOR_SETTINGS

Переменная среды METEOR_SETTINGS может принимать объекты JSON и выставляет этот объект в объекте `Meteor.settings` . Во-первых, добавьте `settings.json` в свой корень приложения с некоторой информацией о конфигурации.

```
{
  "public":{
    "ga":{
      "account":"UA-XXXXXXX-1"
    }
  }
}
```

Затем вам нужно запустить приложение, используя файл настроек.

```
# run your app in local development mode with a settings file
meteor --settings settings.json

# or bundle and prepare it as if you're running in production
# and specify a settings file
meteor bundle --directory /path/to/output
cd /path/to/output
MONGO_URL="mongodb://127.0.0.1:27017" PORT=3000 METEOR_SETTINGS=$(cat /path/to/settings.json)
node main.js
```

Затем эти настройки можно получить из настроек Meteor.settings и использовать в вашем приложении.

```
Meteor.startup(function(){
  if(Meteor.isClient){
    console.log('Google Analytics Account', Meteor.settings.public.ga.account);
  }
});
```

Обнаружение среды на сервере

Переменные среды также доступны для сервера через объект `process.env` .

```
if (Meteor.isServer) {
  Meteor.startup(function () {
    // detect environment by getting the root url of the application
    console.log(JSON.stringify(process.env.ROOT_URL));

    // or by getting the port
    console.log(JSON.stringify(process.env.PORT));

    // alternatively, we can inspect the entire process environment
    console.log(JSON.stringify(process.env));
  });
}
```

Обнаружение среды клиента с использованием методов Meteor

Чтобы обнаружить среду на сервере, мы должны создать вспомогательный метод на сервере, так как сервер определит, в какой среде он находится, а затем вызовите вспомогательный метод от клиента. В принципе, мы просто передаем информацию об окружающей среде с сервера клиенту.

```
//-------------------------------------------------------------------------------------------
----------
// server/server.js
// we set up a getEnvironment method

Meteor.methods({
  getEnvironment: function(){
    if(process.env.ROOT_URL == "http://localhost:3000"){
        return "development";
    }else{
        return "staging";
    }
  }
 });

//-------------------------------------------------------------------------------------------
----------
// client/main.js
// and then call it from the client

Meteor.call("getEnvironment", function (result) {
  console.log("Your application is running in the " + result + "environment.");
});
```

Обнаружение среды клиента с помощью NODE_ENV

Начиная с Meteor 1.3, Meteor теперь предоставляет по `NODE_ENV` переменную `NODE_ENV` на клиенте.

```
if (Meteor.isClient) {
  Meteor.startup(function () {
    if(process.env.NODE_ENV === "testing"){
      console.log("In testing...");
    }
    if(process.env.NODE_ENV === "production"){
      console.log("In production...");
    }
  });
}
```

Прочитайте Обнаружение окружающей среды онлайн:
https://riptutorial.com/ru/meteor/topic/4198/обнаружение-окружающей-среды

глава 35: отладка

Examples

Отладчики браузера

И Chrome, и Safari встроены в отладчики. В Chrome все, что вам нужно сделать, это щелкнуть правой кнопкой мыши на веб-странице и «Проверить элемент». В Safari вам нужно перейти в «Настройки»> «Дополнительно» и нажать «Показать меню« Разработка »в строке меню».

С Firefox вам нужно установить Firebug

Добавить точки отладки для вашего приложения

Вам нужно будет добавить инструкции `debugger` к вашему коду:

```
Meteor.methods({
  doSomethingUself: function(){
    debugger;
    niftyFunction();
  }
});
```

Отладка на стороне сервера с помощью Индекса узлов

Для отладки на стороне сервера вам необходимо использовать инструмент, например Node Inspector. Прежде чем приступить к работе, ознакомьтесь с некоторыми из этих полезных уроков.

HowToNode - отладка с инспектором узлов
Strongloop - приложения для отладки
Легко отлаживать прохождение Meteor.js со скриншотами об использовании Индекса узлов с Метеор

tl; dr - в экосистеме Метеор есть ряд утилит, которые предназначены для запуска одновременно с вашим приложением «Метеор». Они работают только в том случае, если приложение Meteor запущено, и они могут подключаться к работающему веб-сайту. meteor mongo, Robomongo, Nightwatch ... это все утилиты, которые требуют, чтобы ваше приложение уже работало. NodeInspector - одна из этих утилит.

```
# install node-inspector
terminal-a$  npm install -g node-inspector

# start meteor
```

```
terminal-a$  NODE_OPTIONS='--debug-brk --debug' mrt run

# alternatively, some people report this syntax being better
terminal-a$  sudo NODE_OPTIONS='--debug' ROOT_URL=http://myapp.com meteor --port 80

# launch node-inspector along side your running app
terminal-b$  node-inspector

# go to the URL given by node-inspector
http://localhost:8080/debug?port=5858
```

Отладка на стороне сервера с помощью отладки npm

Помимо Node Inspector, некоторые люди сообщили об успехе с утилитой npm, называемой `debug` .

MeteorHacks - отладка Meteor с отладкой npm

Метеорная оболочка

Начиная с Meteor 1.0.2, есть новая командная оболочка, которую вы можете использовать для интерактивной отладки и управления вашим приложением со стороны сервера, так же, как и с консолью Chrome на стороне клиента! Проверьте это:

```
meteor shell
```

Другие утилиты отладки

Метеорный свал
Метеорные игрушки
Созвездие

Meteor DevTools

Прочитайте отладка онлайн: https://riptutorial.com/ru/meteor/topic/3378/отладка

глава 36: Офлайн-приложения

замечания

Дальнейшие исследования Appcache

http://www.html5rocks.com/en/tutorials/indexeddb/todo/
http://grinninggecko.com/2011/04/22/increasing-chromes-offline-application-cache-storage-limit/
http://www.html5rocks.com/en/tutorials/offline/quota-research/
https://developers.google.com/chrome/apps/docs/developers_guide?csw=1#installing
https://developers.google.com/chrome/apps/docs/developers_guide?csw=1#manifest

Examples

Meteor.status ()

Первое, что нужно сделать при отключении вашего приложения Meteor, - это создать визуальную индикацию того, подключено ли локальное клиентское приложение к серверу или нет. Есть много способов сделать это, но самый простой способ - это сделать что-то вроде этого:

```
Template.registerHelper('getOnlineStatus', function(){
  return Meteor.status().status;
});

Template.registerHelper('getOnlineColor', function(){
  if(Meteor.status().status === "connected"){
    return "green";
  }else{
    return "orange";
  }
});
```

```
  <div id="onlineStatus" class="{{getOnlineColor}}">
    {{getOnlineStatus}}
  </div>
```

```
.green{
  color: green;
}
.orange{
  color: orange;
}
```

Включить Appcache

Одним из простых шагов является добавление appcache. Appcache позволит загружать

контент вашего приложения, даже если интернет-доступ отсутствует. Вы не сможете получать какие-либо данные с ваших серверов mongo, но статический контент и активы будут доступны в автономном режиме.

```
meteor add appcache
```

Включить GroundDB

Наконец, мы хотим, чтобы некоторые из наших динамических данных сохранялись в автономном режиме.

```
meteor add ground:db
```

```
Lists = new Meteor.Collection("lists");
GroundDB(Lists);

Todos = new Meteor.Collection("todos")
GroundDB(Todos);
```

Что нужно быть осторожным

- Appcache вызывает некоторую путаницу в вашем рабочем процессе разработки, поскольку он скрывает функции автоматического обновления Meteor. Когда вы отключите серверный компонент приложения, часть клиента в вашем браузере продолжит работу. Это хорошая вещь! Но вы не получаете немедленную обратную связь о том, что ваше приложение отключено, или что были обновления.
- Попробуйте использовать режим Incognito Chrome в процессе разработки приложения, поскольку он не использует appcache.
- GroundDB не работает особенно хорошо с IronRouter.

Прочитайте Офлайн-приложения онлайн: https://riptutorial.com/ru/meteor/topic/3375/офлайн-приложения

глава 37: Переменные среды

параметры

параметр	подробности
PORT	Порт, на котором будет доступно приложение Meteor.
MONGO_URL	URL для подключения к экземпляру Mongo.
ROOT_URL	...
OPLOG_URL	...
MONGO_OPLOG_URL	...
METEOR_ENV	...
NODE_ENV	...
NODE_OPTIONS	...
DISABLE_WEBSOCKETS	...
MAIL_URL	...
DDP_DEFAULT_CONNECTION_URL	...
HTTP_PROXY	...
https_proxy	...
METEOR_OFFLINE_CATALOG	...
METEOR_PROFILE	...
METEOR_DEBUG_BUILD	...
TINYTEST_FILTER	...
MOBILE_ROOT_URL	...
NODE_DEBUG	...
BIND_IP	...

параметр	подробности
PACKAGE_DIRS	...
DEBUG	...
METEOR_PRINT_CONSTRAINT_SOLVER_INPUT	...
METEOR_CATALOG_COMPRESS_RPCS	...
METEOR_MINIFY_LEGACY	...
METEOR_DEBUG_SQL	...
METEOR_WAREHOUSE_DIR	...
AUTOUPDATE_VERSION	...
USE_GLOBAL_ADK	...
METEOR_AVD	...
DEFAULT_AVD_NAME	...
METEOR_BUILD_FARM_URL	...
METEOR_PACKAGE_SERVER_URL	...
METEOR_PACKAGE_STATS_SERVER_URL	...
DEPLOY_HOSTNAME	...
METEOR_SESSION_FILE	...
METEOR_PROGRESS_DEBUG	...
METEOR_PRETTY_OUTPUT	...
APP_ID	...
AUTOUPDATE_VERSION	...
CONSTRAINT_SOLVER_BENCHMARK	...
DDP_DEFAULT_CONNECTION_URL	...
SERVER_WEBSOCKET_COMPRESSION	...
USE_JSESSIONID	...
METEOR_PKG_SPIDERABLE_PHANTOMJS_ARGS	...

параметр	подробности
WRITE_RUNNER_JS	...
TINYTEST_FILTER	...
METEOR_PARENT_PID	...
METEOR_TOOL_PATH	...
RUN_ONCE_OUTCOME	...
TREE_HASH_DEBUG	...
METEOR_DEBUG_SPRINGBOARD	...
METEOR_TEST_FAIL_RELEASE_DOWNLOAD	...
METEOR_CATALOG_COMPRESS_RPCS	...
METEOR_TEST_LATEST_RELEASE	...
METEOR_WATCH_POLLING_INTERVAL_MS	...
EMACS	...
METEOR_PACKAGE_STATS_TEST_OUTPUT	...
METEOR_TEST_TMP	...

Examples

Использование переменных среды с помощью Meteor

Переменные среды могут быть определены до вызова метеора, например:

```
PORT=4000 meteor
NODE_ENV="staging" meteor
```

Настройка SMTP-сервера Meteor

Пример Gmail

```
MAIL_URL=smtp://username%40gmail.com:password@smtp.gmail.com:465/
```

Примечание. Эта настройка позволяет отправлять только 2000 писем в день. См. Https://support.google.com/a/answer/176600?hl=ru для альтернативных конфигураций.

Прочитайте Переменные среды онлайн: https://riptutorial.com/ru/meteor/topic/3154/переменные-среды

глава 38: Полная установка - Mac OSX

Examples

Установить узел и NPM

Этот быстрый старт написан для Mac OSX Mavericks и немного более подробный, чем другие инструкции по установке. Он, надеюсь, должен охватывать несколько краевых случаев, таких как установка вашего пути и настройка NPM, что может привести к сбою установки.

```
    # install node
    # as of OSX Mavericks, we need the GUI installer (?!)
    # when a good command line alternative is found, we'll post it
    http://nodejs.org/download/

    # install npm
    curl -0 -L https://npmjs.org/install.sh | sh

    # check node is installed correctly
    node --version

    # check npm is installed correctly
    npm -version

    # find your npm path
    which npm

    # make sure npm is in your path
    sudo nano ~/.profile
      export PATH=$PATH:/usr/local/bin
```

Прохождение Метеор

Этот быстрый старт написан для Mac OSX Mavericks и немного более подробный, чем другие инструкции по установке. Он, надеюсь, должен охватывать несколько краевых случаев, таких как установка вашего пути и настройка NPM, что может привести к сбою установки.

```
# install meteor
    curl https://install.meteor.com | sh

    # check it's installed correctly
    meteor --version

    # install node
    # as of OSX Mavericks, we need the GUI installer (?!)
    # when a good command line alternative is found, we'll post it
    http://nodejs.org/download/

    # install npm
```

```
    curl -0 -L https://npmjs.org/install.sh | sh

    # check node is installed correctly
    node --version

    # check npm is installed correctly
    npm -version

    # find your npm path
    which npm

    # make sure npm is in your path
    sudo nano ~/.profile
      export PATH=$PATH:/usr/local/bin
```

Монгонская установка

Метеор не существует изолированно, и обычно устанавливается ряд дополнительных инструментов для развития, таких как Mongo, Robomongo, Atom, Linters и т. Д.

```
# make sure mongo is in your local path
nano ~/.profile
  export PATH=$PATH:/usr/local/mongodb/bin

# or install it to the global path
nano /etc/paths
  /usr/local/mongodb/bin

# create mongo database directory
mkdir /data/
mkdir /data/db
chown -R username:admin /data

# run mongodb server
mongod
ctrl-c

# check that you can connect to your meteor app with stand-alone mongo
terminal-a$ meteor create helloworld
terminal-a$ cd helloworld
terminal-a$ meteor

terminal-b$ mongo -port 3001

# install robomongo database admin tool
http://robomongo.org/

# check you can connect to your mongo instance with robomongo
terminal-a$ meteor create helloworld
terminal-a$ cd helloworld
terminal-a$ meteor

Dock$ Robomongo > Create > localhost:3001
```

Другие инструменты для разработки

```
# install node-inspector
terminal-a$  npm install -g node-inspector

# start meteor
terminal-a$  cd helloworld
terminal-a$  NODE_OPTIONS='--debug-brk --debug' mrt run

# alternatively, some people report this syntax being better
terminal-a$  sudo NODE_OPTIONS='--debug' ROOT_URL=http://helloworld.com meteor --port 80

# launch node-inspector along side your running app
terminal-b$  node-inspector

# go to the URL given by node-inspector and check it's running
http://localhost:8080/debug?port=5858

# install jshint
npm install -g jshint
```

Прочитайте Полная установка - Mac OSX онлайн: https://riptutorial.com/ru/meteor/topic/3294/полная-установка---mac-osx

глава 39: Приемочное тестирование (с помощью Nightwatch)

замечания

Nightwatch предоставляет приемные и сквозные тесты для приложений Meteor с v0.5 дней и управляет миграциями от PHP до Spark to Blaze и React; и все основные платформы непрерывной интеграции. Дополнительную помощь можно найти в:

Документация API Nightwatch
Группа пользователей Nightwatch.js Google

Examples

Область приложения

На самом базовом уровне приемочные испытания - это, по сути, тестирование с использованием «черного ящика», которое в основном связано с тестированием входов и выходов замкнутой системы. Таким образом, для приемочного тестирования существуют три существенные особенности: поиск ресурса, чтение данных и запись данных. Когда дело доходит до браузеров и webapps, эти три функции в основном сводятся к следующему:

1. Загрузите представление веб-страницы или приложения
2. Осмотреть элементы пользовательского интерфейса (например, DOM)
3. Запуск события / имитация взаимодействия с пользователем

Мы называем это площадью поверхности приложения. Поверхность - это то, что пользователь видит или испытывает. Это внешняя часть системы Blackbox. А поскольку пользователи взаимодействуют с современными веб-приложениями на видеоэкранах с помощью веб-браузеров, наше покрытие поверхности определяется универсальными локаторами ресурсов (URL-адресами) и видовыми экранами. Итак, наше первое прохождение начинается с того, что выглядит следующим образом:

```
module.exports = {
  "Hello World" : function (client) {
    client
      // the location of our Meteor app
      .url("http://localhost:3000")

      // the size of the viewport
      .resizeWindow(1024, 768)

      // test app output
```

```
      .verify.elementPresent('h1')
      .verify.containsText('h1', "Welcome to Meteor!")
      .verify.containsText('p', "You've pressed the button 0 times")
      .verify.elementPresent('button')

      // simulate user input
      .click('button').pause(500)

      // test app output again, to make sure input worked
      .verify.containsText('p', "button 1 times")

      // saving a copy of our viewport pixel grid
      .saveScreenshot('tests/nightwatch/screenshots/homepage.png')
      .end();
  }
};
```

Пользовательские команды

Nightwatch поддерживает создание пользовательских команд, которые могут имитировать нажатия клавиш, щелчки мыши и другие входы. Пользовательская команда может быть привязана к другим командам Nightwatch, например:

```
module.exports = {
  "Login App" : function (client) {
    client
      .url("http://localhost:3000")
      .login("janedoe@somewhere.com", "janedoe123")
      .end();
  }
};
```

Чтобы включить это, определите команду в `./tests/nightwatch/commands/login` следующим образом:

```
exports.command = function(username, password) {

  this
    .verify.elementPresent('#login')

      // we clear the input in case there's any data remaining from previous visits
      .clearValue("#emailInput")
      .clearValue("#passwordInput")

      // we simulate key presses
      .setValue("#emailInput", username)
      .setValue("#passwordInput", password)

    // and we simulate a mouse click
    .click("#signInToAppButton").pause(1000)

  return this; // allows the command to be chained.
};
```

Чтобы все это работало, вам нужно добавить атрибуты `id` на свою страницу входа. На

каком-то уровне он должен будет примерно выглядеть примерно так:

```
<template name="login">
  <div id="login">
    <input id="emailInput" name="email" type="email" />
    <input id="passwordInput" name="password" type="password" />
    <button id="#signInToAppButton">Sign In</button>
  </div>
</template>
```

Проверка объектов Meteor на клиенте

Поскольку Nightwatch имеет доступ к консоли браузера, можно проверить объекты на стороне клиента, используя API `.execute()` . В следующем примере мы проверяем объект Session для определенной переменной сеанса. Сначала начнем с создания файла `./tests/nightwatch/api/meteor/checkSession` , где мы будем хранить следующую команду:

```
// syncrhonous version; only works for checking javascript objects on client
exports.command = function(sessionVarName, expectedValue) {
  var client = this;
  this
    .execute(function(data){
      return Session.get(data);
    }, [sessionVarName], function(result){
      client.assert.ok(result.value);
      if(expectedValue){
        client.assert.equal(result.value, expectedValue);
      }
    })
    return this;
};
```

Затем мы можем связать его так:

```
module.exports = {
  "Check Client Session" : function (client) {
    client
      .url("http://localhost:3000")
      .checkSession("currentUser", "Jane Doe")
      .end();
  }
};
```

Формы и типы ввода

Чтобы загрузить файл, вам сначала нужно создать каталог / data и добавить файл, который вы хотите загрузить.

```
tests/nightwatch/data/IM-0001-1001.dcm
```

Для вашей формы потребуется ввод с типом файла. (Некоторым людям не нравятся

варианты стилизации, которые предоставляет этот вход, а общий шаблон - сделать этот ввод скрытым, а другой кнопкой на странице щелкнуть его по имени пользователя.)

```
<form id="myform">
    <input type="file" id="fileUpload">
    <input type="text" name="first_name">
    <input type="text" name="last_name">

    <input type="date" name="dob_month">
    <input type="date" name="dob_day">
    <input type="date" name="dob_year">

    <input type="radio" name="gender" value="M">
    <input type="radio" name="gender" value="F">
    <input type="radio" name="gender" value="O">

    <input type="select" name="hs_graduation_year">
    <input type="text" name="city">
    <input type="select" name="state">

    <input type="submit" name="submit" value="Submit">
</form>
```

Затем ваши тесты должны будут использовать setValue () и разрешить путь к локальному файловому ресурсу.

```
module.exports = {
  "Upload Study" : function (client) {
    console.log(require('path').resolve(__dirname +  '/../data' ));

    var stringArray = "Chicago";

    client
      .url(client.globals.url)
      .verify.elementPresent("form#myform")

      // input[type="file"]
      .verify.elementPresent("input#fileUpload")
      .setValue('input#fileUpload', require('path').resolve(__dirname + '/../data/IM-0001-
1001.dcm'))

      // input[type="text"]
      .setValue('input[name="first_name"]', 'First')
      .setValue('input[name="last_name"]', 'Last')

      // input[type="date"]
      .click('select[name="dob_month"] option[value="3"]')
      .click('select[name="dob_day"] option[value="18"]')
      .click('select[name="dob_year"] option[value="1987"]')

      // input[type="radio"]
      .click('input[name="gender"][value="M"]')

      // input[type="number"]
      .click('select[name="hs_graduation_year"] option[value="2002"]')

      // input[type="text"]
      // sometimes Nightwatch will send text faster than the browser can handle
```

```
      // which will cause skipping of letters.  In such cases, we need to slow
      // Nightwatch down; which we do by splitting our input into an array
      // and adding short 50ms pauses between each letter
      for(var i=0; i < userIdArray.length; i++) {
        client.setValue('input[name="city"]', stringArray[i]).pause(50)
      }

      // input[type="select"]
      // after an array input above, we need to resume our method chain...
      client.click('select[name="state"] option[value="CA"]')

      // input[type="number"]
      .setValue('input[name="zip"]', '01234')

      //input [ type="submit" ]
      .click('button[type="submit"]')
      .end();
  }
};
```

Кредит Даниэлю Ринехарту за то, что он проиграл этот пример.

Компоненты и объекты страницы

Объекты страницы аналогичны пользовательским командам; за исключением того, что они представляют собой коллекции пользовательских команд, которые связаны с конкретным компонентом пользовательского интерфейса. Это очень хорошо работает с современным дизайном на основе компонентов, например, в React.

```
module.exports = {
  url: 'http://localhost:3000/login',
  commands: [{
  login: function(email, password) {
    return this
      .clearValue('input[name="emailAddress"]')
      .clearValue('input[name="password"]')

      .setValue('input[name="emailAddress"]', email)
      .setValue('input[name="password"]', password)

      .verify.elementPresent('#loginButton')
      .click("#loginButton");
  },
  clear: function() {
    return this
      .waitForElementVisible('@emailInput')
      .clearValue('@emailInput')
      .clearValue('@passInput')
      .waitForElementVisible('@loginButton')
      .click('@loginButton')
  },
  checkElementsRendered: function(){
    return this
      .verify.elementPresent("#loginPage")
      .verify.elementPresent('input[name="emailAddress"]')
      .verify.elementPresent('input[name="password"]')
  },
```

```
  pause: function(time, client) {
    client.pause(time);
    return this;
  },
  saveScreenshot: function(path, client){
    client.saveScreenshot(path);
    return this;
  }
}],
  elements: {
    emailInput: {
      selector: 'input[name=email]'
    },
    passInput: {
      selector: 'input[name=password]'
    },
    loginButton: {
      selector: 'button[type=submit]'
    }
  }
};
```

Единственное предостережение с использованием шаблона PageObject при тестировании компонентов заключается в том, что реализация прерывает поток цепочки метода, который обеспечивает встроенный `verify.elementPresent` . `verify.elementPresent` . Вместо этого вам нужно назначить объект страницы переменной и создать новую цепочку методов для каждой страницы. Разумная цена за последовательную и надежную модель для повторного использования кода.

```
module.exports = {
  tags: ['accounts', 'passwords', 'users', 'entry'],
  'User can sign up.': function (client) {

    const signupPage = client.page.signupPage();
    const indexPage = client.page.indexPage();

    client.page.signupPage()
      .navigate()
      .checkElementsRendered()
      .signup('Alice', 'Doe', 'alice@test.org', 'alicedoe')
      .pause(1500, client);

    indexPage.expect.element('#indexPage').to.be.present;
    indexPage.expect.element('#authenticatedUsername').text.to.contain('Alice Doe');
  },
}
```

Прочитайте Приемочное тестирование (с помощью Nightwatch) онлайн:
https://riptutorial.com/ru/meteor/topic/6454/приемочное-тестирование--с-помощью-nightwatch-

глава 40: Публикация данных

замечания

В подсистеме данных Meteor публикация сервера и соответствующие подписки клиента являются основными механизмами реактивной передачи данных в реальном времени, когда базовые данные постоянно синхронизируются между сервером и клиентом.

Examples

Основная подписка и публикация

Сначала удалите `autopublish` . `autopublish` автоматически публикует всю базу данных на стороне клиента, поэтому эффекты публикаций и подписки не могут быть видны.

Чтобы удалить `autopublish` :

```
$ meteor remove autopublish
```

Затем вы можете создавать публикации. Ниже приведен полный пример.

```
import { Mongo } from 'meteor/mongo';
import { Meteor } from 'meteor/meteor';

const Todos = new Mongo.Collection('todos');

const TODOS = [
  { title: 'Create documentation' },
  { title: 'Submit to Stack Overflow' }
];

if (Meteor.isServer) {
  Meteor.startup(function () {
    TODOS.forEach(todo => {
      Todos.upsert(
        { title: todo.title },
        { $setOnInsert: todo }
      );
    });
  });

  // first parameter is a name.
  Meteor.publish('todos', function () {
    return Todos.find();
  });
}

if (Meteor.isClient) {
  // subscribe by name to the publication.
  Meteor.startup(function () {
    Meteor.subscribe('todos');
```

```
  })
}
```

Глобальные публикации

Глобальная публикация не имеет имени и не требует подписки от подключенного клиента, и поэтому она доступна подключенному клиенту, как только клиент подключается к серверу.

Чтобы добиться этого, можно просто назвать публикацию `null`

```
Meteor.publish(null, function() {
  return SomeCollection.find();
})
```

Названные публикации

Именованная публикация - это имя, которое имеет имя и должно быть явно подписано клиентом.

Рассмотрим этот код на стороне сервера:

```
Meteor.publish('somePublication', function() {
  return SomeCollection.find()
})
```

Клиент должен запросить его:

```
Meteor.subscribe('somePublication')
```

Подписчики с подменю шаблонов

Система моделирования шаблонов Meteor Spacebars и ее базовая подсистема рендеринга Blaze без видимости интегрируются с методами жизненного цикла публикации, так что простая часть кода шаблона может подписываться на свои собственные данные, останавливать и очищать свои собственные трассы во время разрыва шаблона.

Чтобы воспользоваться этим, нужно подписаться на экземпляр шаблона, а не на символ `Meteor` :

Сначала настройте шаблон

```
<template name="myTemplate">
  We will use some data from a publication here
</template>
```

Затем коснитесь соответствующего обратного вызова жизненного цикла

```
Template.myTemplate.onCreated(function() {
  const templateInstance = this;
  templateInstance.subscribe('somePublication')
})
```

Теперь, когда этот шаблон будет уничтожен, публикация также будет остановлена автоматически.

Примечание. Данные, на которые подписаны, будут доступны для всех шаблонов.

Публикация в эфемерной клиентской стороне.

Ибо, если вам нужно настроить то, что опубликовано.

```
import { Mongo } from 'meteor/mongo';
import { Meteor } from 'meteor/meteor';
import { Random } from 'meteor/random';

if (Meteor.isClient) {
  // established this collection on the client only.
  // a name is required (first parameter) and this is not persisted on the server.
  const Messages = new Mongo.Collection('messages');
  Meteor.startup(function () {
    Meteor.subscribe('messages');
    Messages.find().observe({
      added: function (message) {
        console.log('Received a new message at ' + message.timestamp);
      }
    });
  })
}

if (Meteor.isServer) {
  // this will add a new message every 5 seconds.
  Meteor.publish('messages', function () {
    const interval = Meteor.setInterval(() => {
      this.added('messages', Random.id(), {
        message: '5 seconds have passed',
        timestamp: new Date()
      })
    }, 5000);
    this.added('messages', Random.id(), {
      message: 'First message',
      timestamp: new Date()
    });
    this.onStop(() => Meteor.clearInterval(interval));
  });
}
```

Создание и реагирование на ошибку в публикации.

На сервере вы можете создать такую публикацию. `this.userId` - это идентификатор пользователя, который в настоящий момент вошел в систему. Если ни один пользователь не вошел в систему, вам может потребоваться выбросить ошибку и ответить на нее.

```
import Secrets from '/imports/collections/Secrets';

Meteor.publish('protected_data', function () {
  if (!this.userId) {
    this.error(new Meteor.Error(403, "Not Logged In."));
    this.ready();
  } else {
    return Secrets.find();
  }
});
```

На клиенте вы можете ответить следующим.

```
Meteor.subscribe('protected_data', {
  onError(err) {
    if (err.error === 403) {
      alert("Looks like you're not logged in");
    }
  },
});
```

Файл / импорт / коллекции / Секреты создают ссылку на коллекцию секретов, как показано ниже:

```
const Secrets = new Mongo.Collection('secrets');
```

Реактивировать повторную подписку на публикацию

Автозапуск шаблонов может использоваться для (пере) подписки на публикацию. Он устанавливает реактивный контекст, который повторно выполняется, когда любые *реактивные данные зависят от изменений* . Кроме того, автозапуск всегда выполняется один раз (при первом запуске).

`onCreated` шаблонов обычно помещается в метод `onCreated` .

```
Template.myTemplate.onCreated(function() {
  this.parameter = new ReactiveVar();
  this.autorun(() => {
    this.subscribe('myPublication', this.parameter.get());
  });
});
```

Это будет запускаться один раз (первый раз) и настроить подписку. Затем он будет повторно запускаться всякий раз, когда изменяется переменная `parameter` .

Подождите в режиме Blaze, пока публикуются данные

Шаблон JS-кода

```
Template.templateName.onCreated(function(){
```

```
    this.subscribe('subsription1');
    this.subscribe('subscription2');
});
```

Шаблон HTML-кода

```
<template name="templateName">
    {{#if Template.subscriptionsReady }}
        //your actual view with data. it can be plain HTML or another template
    {{else}}
        //you can use any loader or a simple header
        <h2> Please wait ... </h2>
    {{/if}}
</template>
```

Проверка учетной записи пользователя при публикации

Иногда это хорошая идея для дальнейшей защиты ваших изданий, требуя входа пользователя в систему. Вот как вы достигаете этого через Метеор.

```
import { Recipes } from '../imports/api/recipes.js';
import { Meteor } from 'meteor/meteor';

Meteor.publish('recipes', function() {
  if(this.userId) {
    return Recipe.find({});
  } else {
    this.ready();  // or: return [];
  }
});
```

Публикация нескольких курсоров

Несколько курсоров базы данных могут быть опубликованы из одного метода публикации, возвращая массив курсоров.

Курсоры «дети» будут рассматриваться как объединения и не будут реагировать.

```
Meteor.publish('USER_THREAD', function(postId) {
  let userId   = this.userId;

  let comments = Comments.find({ userId, postId });
  let replies  = Replies.find({ userId, postId });

  return [comments, replies];
});
```

Моделировать задержку публикаций

В реальном мире могут возникнуть задержки соединения и сервера, чтобы имитировать задержки в среде разработки `Meteor._sleepForMs(ms);` может быть использован

```
Meteor.publish('USER_DATA', function() {
    Meteor._sleepForMs(3000); // Simulate 3 seconds delay
    return Meteor.users.find({});
});
```

Объединение публикаций

Публикации могут быть объединены на клиенте, в результате чего документы с разными формами формируются в пределах одного курсора. В следующем примере показано, как каталог пользователя может публиковать минимальное количество общедоступных данных для пользователей приложения и предоставлять более подробный профиль для зарегистрированного пользователя.

```
// client/subscriptions.js
Meteor.subscribe('usersDirectory');
Meteor.subscribe('userProfile', Meteor.userId());

// server/publications.js
// Publish users directory and user profile

Meteor.publish("usersDirectory", function (userId) {
    return Meteor.users.find({}, {fields: {
        '_id': true,
        'username': true,
        'emails': true,
        'emails[0].address': true,

        // available to everybody
        'profile': true,
        'profile.name': true,
        'profile.avatar': true,
        'profile.role': true
    }});
});
Meteor.publish('userProfile', function (userId) {
    return Meteor.users.find({_id: this.userId}, {fields: {
        '_id': true,
        'username': true,
        'emails': true,
        'emails[0].address': true,

        'profile': true,
        'profile.name': true,
        'profile.avatar': true,
        'profile.role': true,

        // privately accessible items, only availble to the user logged in
        'profile.visibility': true,
        'profile.socialsecurity': true,
        'profile.age': true,
        'profile.dateofbirth': true,
        'profile.zip': true,
        'profile.workphone': true,
        'profile.homephone': true,
        'profile.mobilephone': true,
        'profile.applicantType': true
    }});
```

```
});
```

Прочитайте Публикация данных онлайн: https://riptutorial.com/ru/meteor/topic/1323/публикация-данных

глава 41: Развертывание с помощью Upstart

Examples

Служба Upstart

В этом руководстве по развертыванию предполагается, что вы используете сервер Ubuntu и являетесь либо самообслуживанием, либо используете провайдер инфраструктуры как службы (IaaS), например Amazon Web Services или Rackspace. На вашем сервере Ubuntu должен быть запущен демон для запуска других приложений, для которых мы рекомендуем службу Upstart. Вы можете найти больше о Upstart со следующими ссылками:

Upstart - Начало работы
Начало работы с Upstart Scripts на Ubuntu
UbuntuBootupHowTo
Upstart Intro, Cookbook и лучшие практики
Запустите NodeJS как службу на Ubuntu Karmic

Копирование файлов на ваш сервер

Один предпочтительный подход к развертыванию на сервере - использовать Git или GitHub. Это в основном включает в себя вход в ваш сервер, переход в каталог, из которого вы хотите запустить приложение, а затем клонирование ваших файлов непосредственно из GitHub. Затем вы создаете свое приложение на сервере. Такой подход гарантирует, что определенные для платформы файлы создаются правильно, но требует, чтобы Meteor был установлен на сервере (500+ MB), и может привести к созданию немного разных сборок, если ваши серверы немного отличаются.

```
cd /var/www
sudo git clone http://github.com/myaccount/myapp.git
cd /var/www/myapp
meteor build --directory ../myapp-production
sudo service myapp restart
```

Bundle Then Скопировать на сервер

Кроме того, вы можете захотеть создать приложение, а затем развернуть его.

```
cd myapp
meteor build --directory ../output
cd ..
scp output -r username@destination_host:/var/www/myapp-production
```

Написание сценария Upstart

Вам понадобится скрипт upstart в каталоге `/etc/init/ directory` . Назовите его именем вашего приложения, заканчивающимся на `.conf` , например `/etc/init/myapp.conf` . Основной сценарий выскочки выглядит примерно так:

```
## /etc/init/myapp.conf
description "myapp.mydomain.com"
author      "somebody@gmail.com"

# Automatically Run on Startup
start on started mountall
stop on shutdown

# Automatically Respawn:
respawn
respawn limit 99 5

script
    export HOME="/root"
    export MONGO_URL='mongodb://myapp.compose.io:27017/meteor'
    export ROOT_URL='http://myapp.mydomain.com'
    export PORT='80'

    exec /usr/local/bin/node /var/www/myapp/main.js >> /var/log/myapp.log 2>&1
end script
```

Скрипт Upstart для наборов реплик

Если вы используете набор реплик или вам нужно очертить свою базу данных, вам понадобится сценарий выскочки, который выглядит примерно так:

```
# /etc/init/myapp.conf
description "myapp.mydomain.com"
author      "somebody@gmail.com"

# used to be: start on startup
# until we found some mounts weren't ready yet while booting:
start on started mountall
stop on shutdown

# Automatically Respawn:
respawn
respawn limit 99 5

script
    # upstart likes the $HOME variable to be specified
    export HOME="/root"

    # our example assumes you're using a replica set and/or oplog integreation
    export MONGO_URL='mongodb://mongo-a,mongo-b,mongo-c:27017/?replicaSet=meteor'

    # root_url and port are the other two important environment variables to set
    export ROOT_URL='http://myapp.mydomain.com'
    export PORT='80'
```

```
    exec /usr/local/bin/node /var/www/production/main.js >> /var/log/node.log 2>&1
end script
```

Запуск сценария Upstart

Наконец, вам нужно запустить демон Upstart и инициализировать приложение как службу.

```
sudo service myapp start
```

Настройка сервера для размещения нескольких приложений Meteor

https://www.phusionpassenger.com/
https://github.com/phusion/passenger
https://github.com/phusion/passenger/wiki/Phusion-Passenger:-Meteor-tutorial#wiki-installing

Прочитайте Развертывание с помощью Upstart онлайн:
https://riptutorial.com/ru/meteor/topic/3377/развертывание-с-помощью-upstart

глава 42: Реактивный (Vars & Dictionaries)

Examples

Реактивный запрос

Пример кода:

В main.html

```
<template name="test">
     <input type="checkbox" id="checkbox1" name="name" value="data">Check Me
      {{showData}}
</template>
```

В Main.js

```
 var check_status='';
 //Reactive Var Initialization
 Template.main.onCreated(function (){
       check_status=new ReactiveVar({});

 });

 Template.main.helpers({
       showData : function(){
           return Collection.find(check_status.get());
       }
 });

 Template.main.events({
      "change #checkbox1" : function(){
              check_status.set({field: 'data'});
       }
 });
```

Объяснение:

Когда мы инициализируем реактивный var `check_status` мы устанавливаем значение, равное `{}` . В помощнике во время рендеринга те же данные передаются в запрос `Collection.find(check_status.get())` который так же хорош, как и *отображение всех* данных.

Как только вы нажмете на флажок, запускается событие, описанное в `Template.main.events` которое устанавливает значение `check_status` в `{field: data}` . Так как это *реактивный var* , шаблон `showData` снова запускается, и на этот раз запрос - `Collection.find({field: data})` , поэтому только поля, где возвращается `field` сопоставляемое `'data'` .

Перед использованием этих команд вам необходимо добавить `reactive var` пакет `reactive var` (`meteor add reactive-var`).

Прочитайте Реактивный (Vars & Dictionaries) онлайн:
https://riptutorial.com/ru/meteor/topic/6535/реактивный--vars--amp--dictionaries-

глава 43: Рецепты пользовательского интерфейса Blaze (Bootstrap, No jQuery)

замечания

Вышеупомянутые примеры Blaze очень совместимы с библиотекой http://bootsnipp.com/ , которая предоставляет только HTML и CSS для компонентов и оставляет javascript до разработчика. Это позволяет компонентам совместно использовать одни и те же методы сортировки, фильтрации, запроса и курсора.

Examples

Выпадающее меню

В следующем примере создается раскрывающееся меню Bootstrap, используя только Blaze и JQuery.

Объектная модель документа

```
    <nav class="nav navbar-nav">
      <li class="dropdown">
        <a href="#" class="dropdown-toggle" data-toggle="dropdown">{{getSelectedValue}} <span
class="glyphicon glyphicon-user pull-right"></span></a>
        <ul class="fullwidth dropdown-menu">
          <li id="firstOption" class="fullwidth"><a href="#">15 Minutes <span class="glyphicon
glyphicon-cog pull-right"></span></a></li>
          <li class="divider"></li>
          <li id="secondOption"><a href="#">30 Minutes <span class="glyphicon glyphicon-stats
pull-right"></span></a></li>
          <li class="divider"></li>
          <li id="thirdOption"><a href="#">1 Hour <span class="badge pull-right"> 42
</span></a></li>
          <li class="divider"></li>
          <li id="fourthOption"><a href="#">4 Hour <span class="glyphicon glyphicon-heart
pull-right"></span></a></li>
          <li class="divider"></li>
          <li id="fifthOption"><a href="#">8 Hours <span class="glyphicon glyphicon-log-out
pull-right"></span></a></li>
        </ul>
      </li>
    </nav>
```

Javascript

```
Template.examplePage.helpers({
  getSelectedValue:function(){
```

```
    return Session.get('selectedValue');
  }
});
Template.dropDownWidgetName.events({
  'click #firstOption':function(){
    Session.set('selectedValue', 1);
  },
  'click #secondOption':function(){
    Session.set('selectedValue', "blue");
  },
  'click #thirdOption':function(){
    Session.set('selectedValue', $('#thirdOption').innerText);
  },
  'click #fourthOption':function(){
    Session.set('selectedValue', Session.get('otherValue'));
  },
  'click #fifthOption':function(){
    Session.set('selectedValue', Posts.findOne(Session.get('selectedPostId')).title);
  },
});
```

Navbars

Очень общая задача - создать отзывчивые навигаторы и создать панели действий / нижнего колонтитула, которые имеют разные элементы управления на основе того, на какой странице находится пользователь, или на какую роль принадлежит пользователь. Давайте рассмотрим, как сделать эти элементы управления.

маршрутизатор

```
Router.configure({
  layoutTemplate: 'appLayout',
});
Router.route('checklistPage', {
    path: '/lists/:_id',
    onBeforeAction: function() {
      Session.set('selectedListId', this.params._id);
      this.next();
    },
    yieldTemplates: {
      'navbarFooter': {
        to: 'footer'
      }
    }
  });
```

Создание шаблона Navbar

```
<template name="navbarFooter">
  <nav id="navbarFooterNav" class="navbar navbar-default navbar-fixed-bottom"
role="navigation">
    <ul class="nav navbar-nav">
      <li><a id="addPostLink"><u>A</u>dd Post</a></li>
      <li><a id="editPostLink"><u>E</u>dit Post</a></li>
      <li><a id="deletePostLink"><u>D</u>elete Post</a></li>
    </ul>
```

```
    <ul class="nav navbar-nav navbar-right">
      <li><a id="helpLink"><u>H</u>elp</a></li>
    </ul>
  </nav>
</template>
```

Определение доходности в макете

```
<template name="appLayout">
  <div id="appLayout">
    <header id="navbarHeader">
      {{> yield 'header'}}
    </header>

      <div id="mainPanel">
        {{> yield}}
      </div>

    <footer id="navbarFooter" class="{{getTheme}}"">
      {{> yield 'footerActionElements' }}
    </footer>
  </div>
</template>
```

модальности

Этот подход является методом чистого Blaze для переключения элементов пользовательского интерфейса в и из существования. Подумайте об этом как о замене модальных диалогов. На самом деле существует множество способов реализации модальных диалогов с использованием этого метода (просто добавьте фоновые маски и анимации).

Объектная модель документа

```
<template name="topicsPage">
  <div id="topicsPage" class="container">
    <div class="panel">
      <div class="panel-heading">
        Nifty Panel
      </div>
      <!-- .... -->
      <div class="panel-footer">
        <!-- step 1.  we click on the button object -->
        <div id="createTopicButton" class="btn {{ getPreferredButtonTheme }}">Create
Topic</div>
      </div>
    </div>

    <!-- step 5 - the handlebars gets activated by the javascript controller -->
    <!-- and toggle the creation of new objects in our model -->
    {{#if creatingNewTopic }}
    <div>
      <label for="topicTextInput"></label>
      <input id="topicTextInput" value="enter some text..."></input>
      <button class="btn btn-warning">Cancel</button>
```

```
      <button class="btn btn-success">OK</button>
    </div>
    {{/if}}
  </div>
</template>
```

Javascript

```
// step 2 - the button object triggers an event in the controller
// which toggles our reactive session variable
Template.topicsPage.events({
  'click #createTopicButton':function(){
    if(Session.get('is_creating_new topic'){
      Session.set('is_creating_new_topic', false);
    }else{
      Session.set('is_creating_new_topic', true);
    }
  }
});

// step 0 - the reactive session variable is set false
Session.setDefault('is_creating_new_topic', false);

// step 4 - the reactive session variable invalidates
// causing the creatNewTopic function to be rerun
Template.topicsPage.creatingNewTopic = function(){
  if(Session.get('is_creating_new_topic')){
    return true;
  }else{
    return false;
  }
}
```

Tagging

Уровень базы данных Сначала мы хотим настроить протокол распространения данных, чтобы мы могли сохранять данные в базе данных и получать их клиенту. Необходимо создать три файла ... один на сервере, один на клиенте и один, который разделяют между ними.

```
// client/subscriptions.js
Meteor.subscribe('posts');

//lib/model.js
Posts  = new Meteor.Collection("posts");
Posts.allow({
    insert: function(){
        return true;
    },
    update: function () {
        return true;
    },
    remove: function(){
        return true;
    }
});
```

```
// server.publications.js
Meteor.publish('posts', function () {
  return Posts.find();
});
```

В этом примере предполагается следующая схема документа для шаблона тегов:

```
{
  _id: "3xHCsDexdPHN6vt7P",
  title: "Sample Title",
  text: "Lorem ipsum, solar et...",
  tags: ["foo", "bar", "zkrk", "squee"]
}
```

Объектная модель документа

Во-вторых, мы хотим создать нашу объектную модель в прикладном уровне. Ниже приведен пример использования панели Bootstrap для рендеринга сообщения с заголовком, текстом и тегами. Обратите внимание, что `selectedPost` , `tagObjects` и `tag` являются вспомогательными функциями шаблона blogPost. `title` и `text` являются полями из нашей записи документа.

```
<template name="blogPost">
  {{#with selectedPost }}
    <div class="blogPost panel panel-default">
      <div class="panel-heading">
        {{ title }}
      </div>
        {{ text }}
      <div class="panel-footer">
        <ul class="horizontal-tags">
          {{#each tagObjects }}
          <li class="tag removable_tag">
            <div class="name">{{tag}}<i class="fa fa-times"></i></div>
          </li>
          {{/each}}
          <li class="tag edittag">
            <input type="text" id="edittag-input" value="" /><i class="fa fa-plus"></i>
          </li>
        </ul>
      </div>
    </div>
  {{/with}}
</template>
```

Javascript

Затем мы хотим настроить некоторые контроллеры для возврата данных, внедрения некоторых данных и т. Д.

```
// you will need to set the selectedPostId session variable
// somewhere else in your application
Template.blogPost.selectedPost = function(){
  return Posts.findOne({_id: Session.get('selectedPostId')});
```

```
}

// next, we use the _.map() function to read the array from our record
// and convert it into an array of objects that Handlebars/Spacebars can parse
Template.blogPost.tagObjects = function () {
    var post_id = this._id;
    return _.map(this.tags || [], function (tag) {
        return {post_id: post_id, tag: tag};
    });
};

// then we wire up click events
Template.blogPost.events({
    'click .fa-plus': function (evt, tmpl) {
        Posts.update(this._id, {$addToSet: {tags: value}});
    },
    'click .fa-times': function (evt) {
        Posts.update({_id: this._id}, {$pull: {tags: this.tag}});
    }
});
```

стайлинг

Наконец, мы хотим определить несколько разных представлений для телефона, планшета и настольных компьютеров; и некоторый базовый стиль пользовательского интерфейса в зависимости от ввода пользователя. В этом примере используется прекомпилятор Less, хотя синтаксис должен быть примерно одинаковым для Sass и Stylus.

```
// default desktop view
.fa-plus:hover{
  cursor: pointer;
}
.fa-times:hover{
  cursor: pointer;
}
// landscape orientation view for tablets
@media only screen and (min-width: 768px) {
  .blogPost{
     padding: 20px;
  }
}
// portrait orientation view for tablets
@media only screen and (max-width: 768px) {
  .blogPost{
     padding: 0px;
       border: 0px;
  }
}
// phone view
@media only screen and (max-width: 480px) {
  blogPost{
   .panel-footer{
       display: none;
    }
  }
}
```

Оповещения и ошибки

Оповещения и ошибки - это самый простой из всех моделей компонентов Meteor. На самом деле они настолько просты, что они едва ли регистрируются как образец сами по себе. Вместо добавления модулей или шаблонов FlashAlert все, что вам действительно нужно сделать, это стиль подходящего шаблона руля, добавьте помощника и подключите его к реактивной переменной сеанса.

Предпосылки
Следующий код требует прекомпилятора LESS и Bootstrap-3 соответственно. Вам нужно будет запустить следующие команды в командной строке, чтобы заставить их работать.

```
meteor add less
meteor add ian:bootstrap-3
```

Объектная модель документа: определение объекта предупреждения Начните с добавления некоторых элементов в объектную модель документа. В этом случае мы хотим создать элемент div для нашего оповещения, который подключен до двух помощников Handlebar.

```
<template name="postsPage">
  <div id="postsPage" class="page">
    <div id="postsPageAlert" class="{{alertColor}}">{{alertMessage}}</div>
    <div class="postsList">
      <!-- other code you can ignore in this example -->
    </div>
    <div id="triggerAlertButton" class="btn btn-default">
  </div>
</template>
```

Javascript: определение помощников шаблонов. Затем мы хотим подключить некоторые контроллеры, которые будут заполнять объектную модель данными. Мы делаем это с двумя реактивными переменными сеанса и двумя помощниками руля.

```
Session.setDefault('alertLevel', false);
Session.setDefault('alertMessage', "");

Template.postsPage.alertColor = function(){
 if(Session.get('alertLevel') == "Success"){
  return "alert alert-success";
 }else if(Session.get('alertLevel') == "Info"){
  return "alert alert-info";
 }else if(Session.get('alertLevel') == "Warning"){
  return "alert alert-warning";
 }else if(Session.get('alertLevel') == "Danger"){
  return "alert alert-danger";
 }else{
  return "alert alert-hidden"
 }
}

Template.postsPage.alertMessage = function(){
  return Session.get('alertMessage');
}
```

Styling: определение видимости DOM. Затем мы хотим вернуться к нашему CSS и определить два представления элемента postsPage. В первом представлении мы отображаем все содержимое нашей объектной модели. Во втором представлении отображается только часть содержимого нашей объектной модели.

```
#postsPage{
  .alert{
    display: block;
  }
  .alert-hidden{
    display: none;
  }
}
```

Javascript: запуск оповещения
Наконец, мы возвращаемся к нашим контроллерам, и мы определяем контроллер событий, который вызывает наше предупреждение при нажатии.

```
Template.postsPage.events({
  'click #triggerAlertButton':function(){
    Session.set('alertLevel', 'Success');
    Session.set('alertMessage', 'You successfully read this important alert message.');
  }
});
```

И это все! Супер простой, не так ли? Теперь вы можете установить переменные сеанса `alertLevel` и `alertMessage` любом месте вашей кодовой базы, и ваше приложение будет реагировать на предупреждения и сообщения об ошибках! :)

Рабочий процесс с вкладками

Объектная модель документа
Начните с создания вкладок и панелей в объектной модели ...

```
<template name="samplePage">
  <div id="samplePage" class="page">
    <ul class="nav nav-tabs">
      <li id="firstPanelTab"><a href="#firstPanel">First</a></li>
      <li id="secondPanelTab"><a href="#secondPanel">Second</a></li>
    </ul>

    <div id="firstPanel" class="{{firstPanelVisibility}}">
      {{> firstPanel }}
    </div>
    <div id="secondPanel" class="{{secondPanelVisibility}}">
      {{> secondPanel }}
    </div>
  </div>
</template>
```

Javascript

```
// this variable controls which tab is displayed and associated application state
Session.setDefault('selectedPanel', 1);

Template.name.helpers({
  firstPanelVisibility: function (){
    if(Session.get('selectedPanel') === 1){
      return "visible";
    }else{
      return "hidden";
    }
  },
  secondPanelVisibility: function (){
    if(Session.get('selectedPanel') === 2){
      return "visible";
    }else{
      return "hidden";
    }
  },
  thirdPanelVisibility: function (){
    if(Session.get('selectedPanel') === 3){
      return "visible";
    }else{
      return "hidden";
    }
  },
  firstPanelActive: function (){
    if(Session.get('selectedPanel') === 1){
      return "active panel-tab";
    }else{
      return "panel-tab";
    }
  },
  secondPanelActive: function (){
    if(Session.get('selectedPanel') === 2){
      return "active panel-tab";
    }else{
      return "panel-tab";
    }
  },
  thirdPanelActive: function (){
    if(Session.get('selectedPanel') === 3){
      return "active panel-tab";
    }else{
      return "panel-tab";
    }
  }
});
```

стайлинг

```
.visible {
  display: block;
  visibility: visible;
}
.hidden {
  display: none;
  visibility: hidden;
}
```

Активная вкладка. Для дополнительного эффекта вы можете расширить этот шаблон, введя классы, чтобы указать активную вкладку.

```
<li id="firstPanelTab" class="{{firstPanelActive}}"><a href="#firstPanel">First</a></li>
<li id="secondPanelTab" class="{{secondPanelActive}}"><a href="#secondPanel">Second</a></li>
```

```
Template.firstPanel.helpers({
  firstPanelActive: function (){
    if(Session.get('selectedPanel') === 1){
      return "active";
    }else{
      return "";
    }
  },
  secondPanelActive: function (){
    if(Session.get('selectedPanel') === 2){
      return "active";
    }else{
      return "";
    }
  },
});
```

Прочитайте Рецепты пользовательского интерфейса Blaze (Bootstrap, No jQuery) онлайн: https://riptutorial.com/ru/meteor/topic/4202/рецепты-пользовательского-интерфейса-blaze--bootstrap--no-jquery-

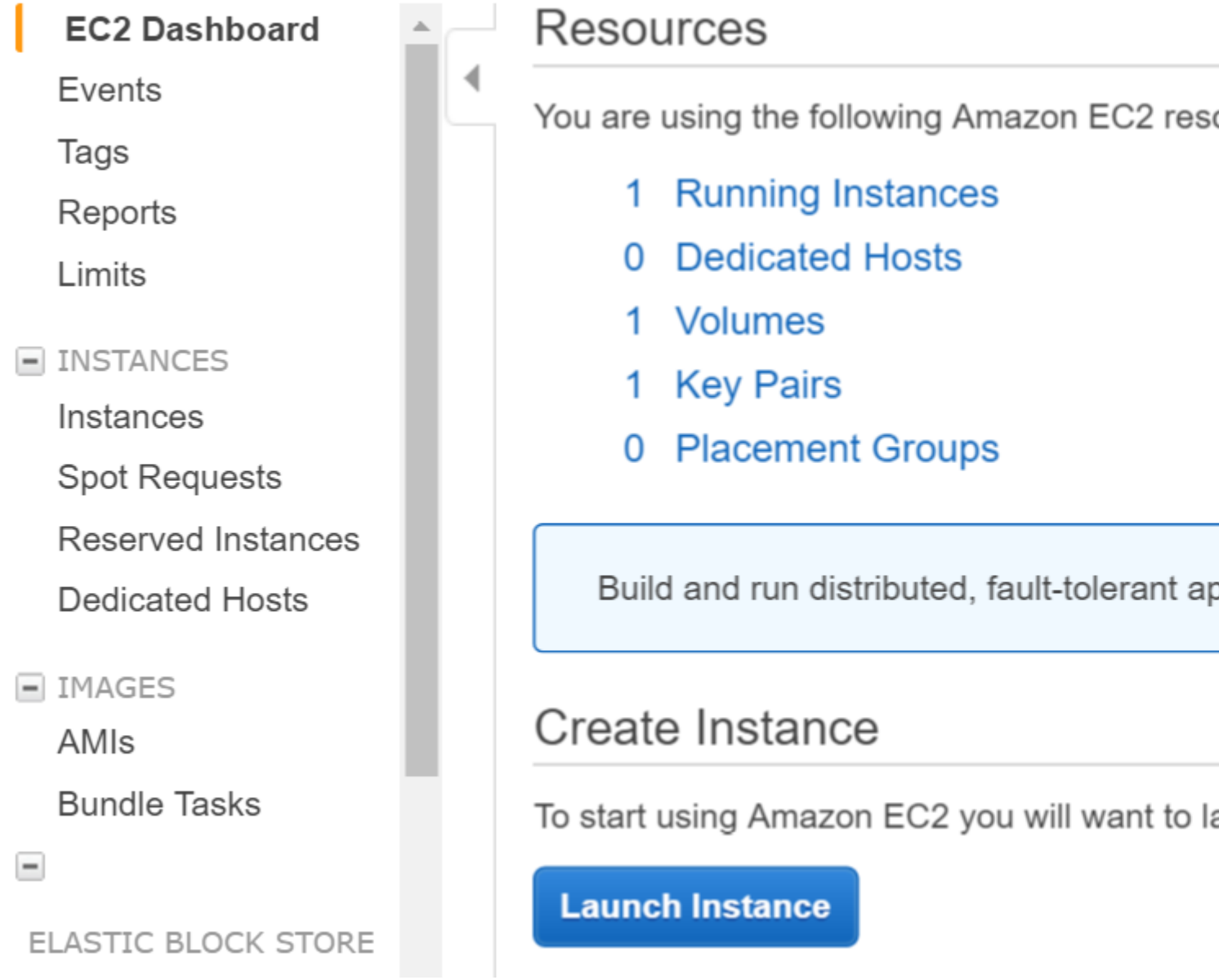
глава 44: Руководство для начинающих по установке Meteor 1.4 на AWS EC2

Examples

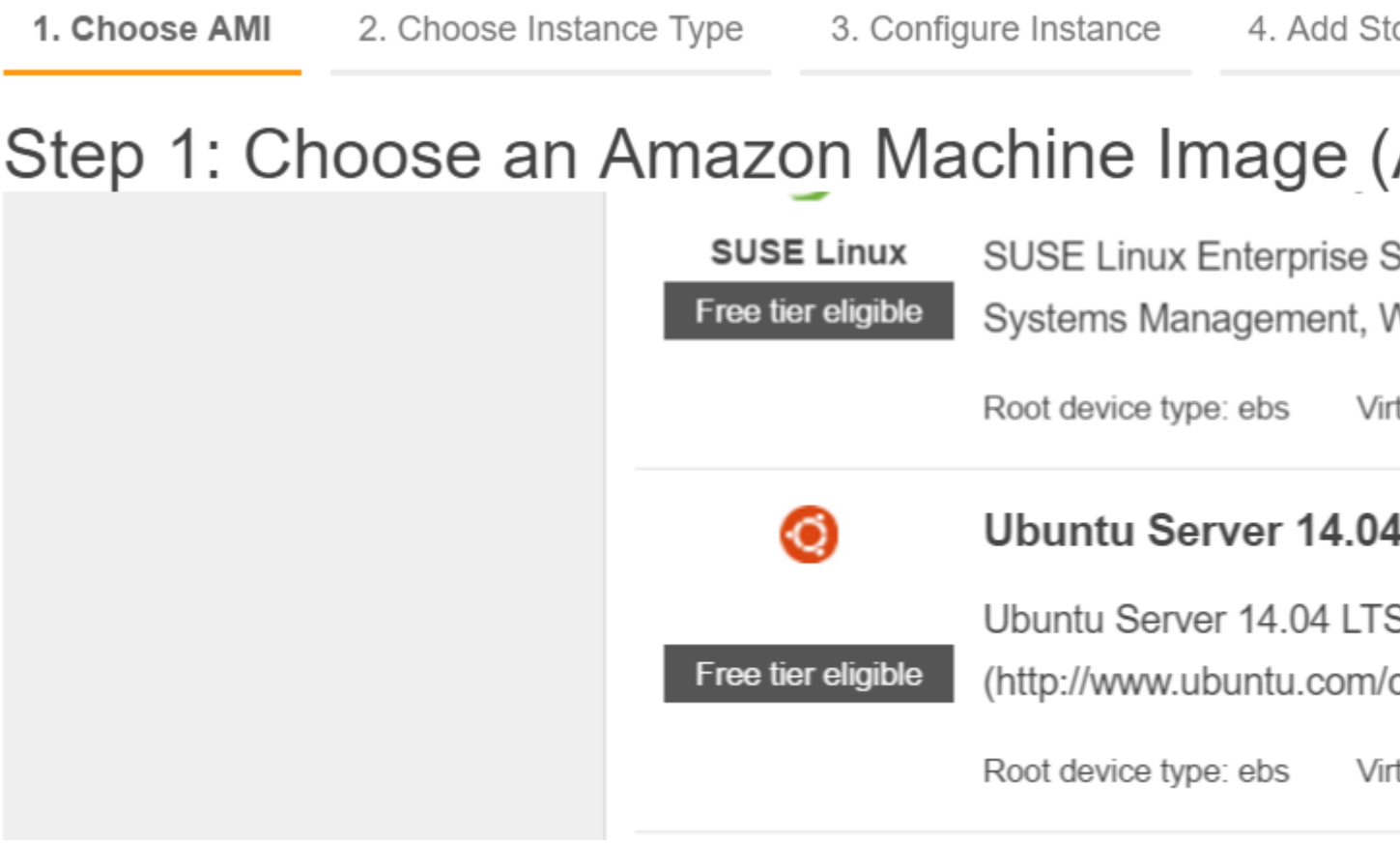
Регистрация для службы AWS

Поскольку много новичков путают о облачном хостинге. Я пишу это руководство, чтобы пройти через установку метеора на aws с помощью ubuntu os. Если вы уже используете свой экземпляр, не стесняйтесь пропустить этот шаг и перейти прямо к установке meteor на aws.

Войдите в консоль AWS. Выберите EC2. Перейдите на панель инструментов EC2. В разделе Создать экземпляр запуска экземпляра экземпляра.



Выберите экземпляр ubuntu на следующем шаге



Создайте ключную пару и загрузите закрытый ключ на свой локальный компьютер.

Войдите через оболочку в aws (используя закрытый ключ, убедитесь, что закрытый ключ находится в вашем пути или запускает команду из каталога, содержащего закрытый ключ)

```
ssh -i "myprivatekey.pem" ubuntu@ec2-xx-xx-xx-xx.ap-south-1.compute.amazonaws.com
```

ec2-xx-xx-xx-xx.ap-south-1.compute.amazonaws.com - это имя экземпляра публичного dns на консоли amazon. ubuntu - имя пользователя. Вы также можете использовать общедоступный IP-адрес.

ШАГИ ДЛЯ УСТАНОВКИ METEOR ON AWS INSTANCE (с использованием mupx)

1. Скопировать секретный ключ из локальной машины в папку aws server ssh

example `/home/ubuntu/.ssh/myprivatekey.pem`

2. обновить упаковщик до последней версии

```
sudo apt-get update
```

3. установить свойства программного обеспечения python

```
 sudo apt-get install python-software-properties
```

4. установить npm и узел (возможно, также установить nvm)

```
sudo apt-get install npm
```

Установить nvm

```
curl https://raw.githubusercontent.com/creationix/nvm/v0.11.1/install.sh | bash
```

Установить узел

```
nvm install 4.4.7

nvm use 4.4.7
```

5. Установить aws cli

```
sudo apt-get install awscli
```

6. Установите метеорит вверх

```
 sudo npm install -g mupx

 sudo npm install -g mupx-letsencrypt
```

(meteor 1.4 в настоящее время доступен только mpux-letencrypt)

7. Инициализировать mupx, перейдя в каталог проекта или создайте новый каталог, если он не существует

```
mupx-letsencrypt init
```

Если вы получите ошибку, как показано ниже, то может существовать устаревший узел, вам нужно создать ссылку

```
/usr/bin/env: node: No such file or directory



sudo ln -s /usr/bin/nodejs /usr/bin/node
```

8. Установить метеор

```
curl https://install.meteor.com | /bin/sh
```

9. edit mup.json (Обязательно заполните имя пользователя: ubuntu и правильное

расположение закрытого ключа с шага 1)

использовать редактор файлов nano (редактировать файлы на ubuntu, также можно использовать vi)

```
nano mup.json
```

Пример mup.json

```
    {
  // Server authentication info
  "servers": [
    {
      "host": "ec2-xx-xx-xx-xx.ap-south-1.compute.amazonaws.com",
      "username": "ubuntu",
      //"password": "password",
      // or pem file (ssh based authentication)
      "pem": "~/.ssh/myprivatekey.pem",
      // Also, for non-standard ssh port use this
      //"sshOptions": { "port" : 49154 },
      // server specific environment variables
      "env": {}
    }
  ],

  // Install MongoDB on the server. Does not destroy the local MongoDB on future setups
  "setupMongo": true,

  // WARNING: Node.js is required! Only skip if you already have Node.js installed on server.
  "setupNode": false,

  // WARNING: nodeVersion defaults to 0.10.36 if omitted. Do not use v, just the version
number.
  //"nodeVersion": "4.4.7",

  // Install PhantomJS on the server
  "setupPhantom": true,

  // Show a progress bar during the upload of the bundle to the server.
  // Might cause an error in some rare cases if set to true, for instance in Shippable CI
  "enableUploadProgressBar": true,

  // Application name (no spaces).
  "appName": "my-app",

  // Location of app (local directory). This can reference '~' as the users home directory.
  // i.e., "app": "/Users/ubuntu/my-app",
  // This is the same as the line below.
  "app": "/Users/ubuntu/my-app",

  // Configure environment
  // ROOT_URL must be set to https://YOURDOMAIN.com when using the spiderable package & force
SSL
  // your NGINX proxy or Cloudflare. When using just Meteor on SSL without spiderable this is
not necessary
  "env": {
    "PORT": 80,
    "ROOT_URL": "http://myapp.com",
```

```
     // only needed if mongodb is on separate server
    "MONGO_URL": "mongodb://url:port/MyApp",
    "MAIL_URL":   "smtp://postmaster%40myapp.mailgun.org:adj87sjhd7s@smtp.mailgun.org:587/"
  },

  // Meteor Up checks if the app comes online just after the deployment.
  // Before mup checks that, it will wait for the number of seconds configured below.
  "deployCheckWaitTime": 60
}
```

10. Установите Meteor, включая mongo, выполнив следующую команду в каталоге проекта.

    ```
    mupx-letsencrypt setup
    ```

11. развертывание проекта с использованием mupx

    ```
    mupx-letsencrypt deploy
    ```

Некоторые полезные команды

Чтобы проверить журналы mupx

```
mupx logs -f
```

Чтобы проверить Docker

```
 docker -D info
```

Проверка состояния сети

```
netstat -a
```

Чтобы проверить текущий текущий процесс, включая использование процессора и памяти

```
 top
```

Установите клиент mongo, чтобы получить доступ к оболочке монго на aws

```
sudo apt-get install mongodb-clients
```

Выполнение запросов mongodb

```
mongo projectName
```

После того, как внутри оболочки манго

```
db.version()
db.users.find()
```

Спасибо arunoda за предоставление замечательного инструмента https://github.com/arunoda/meteor-up

Спасибо команде mupx-letencrypt за хорошую работу. https://www.npmjs.com/package/mupx-letsencrypt

Прочитайте Руководство для начинающих по установке Meteor 1.4 на AWS EC2 онлайн: https://riptutorial.com/ru/meteor/topic/4773/руководство-для-начинающих-по-установке-meteor-1-4-на-aws-ec2

глава 45: Структура каталога

Вступление

Перед выпуском Meteor 1.3 разработчики Meteor были разочарованы обработкой Meteor.js файловых зависимостей и глобальных переменных. В ответ Метеор установил новые стандарты для проектных структур, чтобы сделать систему зависимостей проекта более упорядоченной. В этом разделе объясняется стандартизованная структура проекта и принципы, лежащие в его основе.

замечания

клиент
Весь код в каталоге клиента запускается только на стороне клиента или в веб-браузере.

клиент / совместимость
Каталог совместимости содержит устаревший или сторонний код, например библиотеки jQuery и т. Д.

Lib
Каталог lib загружается перед другими каталогами в вашем проекте Meteor и загружается как на сервере, так и на клиента. Это предпочтительное место для определения моделей данных, изоморфных библиотек и бизнес-логики.

импорт
Каталог импорта - это каталог на сервере, доступный как для сервера, так и для клиента, но только до того, как клиентский пакет будет отправлен клиенту.

пакеты
Каталог пакетов - это где пользовательские пакеты хранятся во время локальной разработки. При использовании стандартной команды `meteor add package:name` для добавления пакета Meteor будет выглядеть первым в этом каталоге, если локальный пакет имеет соответствующее имя описания в файле `package.js` . Если нет, он будет опросить Атмосферу как обычно.

частный
Частный каталог содержит статические файлы, которые должны быть доступны только на веб-сервере.

общественности
Общий каталог содержит статические файлы, доступные только на клиенте приложения. Это может включать в себя активы брендинга и т. Д.

сервер
Каталог сервера содержит серверные активы. Это может включать логику аутентификации, методы и другой код, которые могут потребовать рассмотрения безопасности.

тесты
Каталоги тестов опущены по умолчанию, когда ваше приложение в комплекте и развернуто.

Как было предложено Ричардом Сильвертоном , удобной идеей является размещение не только каталога проекта метеоров под контролем версий, но и его родительского каталога.

Таким образом, вы можете хранить файлы под контролем версий, не имея метеор для борьбы с ним.

Examples

Классические структуры каталогов

Первое, что вам нужно знать при структурировании ваших приложений, - это то, что инструмент Meteor имеет некоторые каталоги, жестко закодированные с определенной логикой. На самом базовом уровне следующие каталоги «запекаются» в пакете «Метеор».

```
client/                                 # client application code
client/compatibility/                   # legacy 3rd party javascript libraries
imports/                                # for lazy loading feature
lib/                                    # any common code for client/server.
packages/                               # place for all your atmosphere packages
private/                                # static files that only the server knows about
public/                                 # static files that are available to the client
server/                                 # server code
tests/                                  # unit test files (won't be loaded on client or
server)
```

Справочная страница: Руководство Meteor> Специальные каталоги

Структура каталога только для пакета

Многие люди оказываются в конечном итоге поддержкой нескольких приложений и хотят поделиться кодом между приложениями. Это приводит к концепции архитектуры микросервиса и приложений для всех пакетов. По сути, код из всей классической структуры каталогов реорганизуется в пакеты.

Несмотря на отсутствие жестко закодированной логики для каталогов в пакетах, мы обнаруживаем, что при создании пакетов рекомендуется использовать классическую структуру каталогов. Это создает естественный путь рефакторинга, поскольку функции

прототипируются в приложении, а затем извлекаются в пакеты, которые должны публиковаться и совместно использоваться. Имена каталогов разделяются, поэтому среди членов команды меньше путаницы.

```
client/                                 # client application code
packages/                               # place for all your atmosphere packages
packages/foo/client                     # client application code
packages/foo/lib                        # any common code for client/server
packages/foo/server                     # server code
packages/foo/tests                      # tests
server/                                 # server code
```

Структура каталогов импорта / модулей

Самые последние версии корабля Метеор с поддержкой `ecmascript` , иначе ES6 или ES2015. Вместо пакетов Javascript теперь поддерживает `import` и модули, что заменяет необходимость в приложениях только для пакетов. Последняя структура каталогов похожа на структуру только для пакетов, но использует каталог `/imports` вместо `/packages` .

```
imports                                 #
imports/api                             # isomorphic methods
imports/lib                             # any common code for client/server
imports/client                          # client application code
imports/server                          # server code
```

Структура каталога смешанного режима

И, конечно же, вы можете смешивать эти подходы и использовать как пакеты, так и импорт вдоль вашего кода, специфичного для вашего приложения. Структура смешанного режима наиболее распространена в трех ситуациях: приложение franken, которое просто немного отвлекает отсюда - и - там без какой-либо общей стратегии; приложение, которое активно реорганизуется из структур Classic или Package-Only в структуру Imports / Modules.

```
client/                                 # client application code
client/compatibility/                   # legacy 3rd party javascript libraries
imports                                 #
imports/api                             # isomorphic methods
imports/lib                             # any common code for client/server
imports/client                          # client application code
imports/server                          # server code
lib/                                    # any common code for client/server.
packages/                               # place for all your atmosphere packages
packages/foo/client                     # client application code
packages/foo/lib                        # any common code for client/server
packages/foo/server                     # server code
packages/foo/tests                      # tests
private/                                # static files that only the server knows about
public/                                 # static files that are available to the client
server/                                 # server code
tests/                                  # unit test files (won't be loaded on client or
server)
```

Порядок загрузки каталога

Файлы шаблонов HTML всегда загружаются перед всем остальным

Файлы, начинающиеся с *основного.* загружаются последними

Затем загружаются файлы внутри любого каталога lib /

Затем загружаются файлы с более глубокими путями

Затем файлы загружаются в алфавитном порядке по всему пути

Ссылка ссылки

Справочная страница: Руководство Meteor> Структура приложения> Порядок загрузки файлов по умолчанию

Прочитайте Структура каталога онлайн: https://riptutorial.com/ru/meteor/topic/3072/структура-каталога

глава 46: Узел / НМП

Examples

Протестированная / поддерживаемая версия узла Meteor

Чтобы определить последнюю тестируемую / поддерживаемую версию узла, которая может быть использована с установленной вами версией Meteor, выгрузите версию узла непосредственно из экземпляра узла узла сборки.

```
meteor node -v
```

Прочитайте Узел / НМП онлайн: https://riptutorial.com/ru/meteor/topic/4599/узел---нмп

глава 47: Управление базой данных Mongo

замечания

Если вы не против использования утилиты Ruby, Genghis является классикой:
http://genghisapp.com/

Но для масштабируемого использования продукции, займитесь MongoHQ.
http://www.mongohq.com/

Кроме того, Mongo Monitoring Service, от 10Gen, создатели Монго:
https://mms.mongodb.com/

MongoClient написан в Meteor, совершенно бесплатно, с открытым исходным кодом и кросс-платформой.

RoboMongo Родной кросс-платформенный инструмент управления MongoDB

Examples

Анализ наследуемой базы данных

Существует две замечательные утилиты для анализа баз данных с использованием «черного ящика». Прежде всего, это multi.js, который даст вам обзор на высоком уровне. Вторая - schema.js, которая позволит вам копаться в коллекциях для более подробной информации о отдельных полях. При наследовании базы данных Mongo производства эти две утилиты могут помочь вам понять, что происходит и как структурируются коллекции и документы.

variety.js

```
mongo test --eval "var collection = 'users'" variety.js
```

schema.js

```
mongo --shell schema.js
```

Подключиться к базе данных на * .meteorapp.com

Флаг -url может оказаться сложным для использования. Для аутентификации есть 60-секундное окно, а затем имя пользователя / пароль случайно сбрасывается. Поэтому убедитесь, что robomongo открыт и готов настроить новое соединение при запуске команды.

```
# get the MONGO_URL string for your app
meteor mongo --url $METEOR_APP_URL
```

Загрузите базу данных из * .meteor.com

То же, что и раньше, но вам нужно скопировать информацию в команду mongodump. Вам необходимо быстро выполнить следующие команды, и для этого требуется координация рук / глаз. Имейте в виду! Это смехотворно хаки! Но весело! Подумайте об этом как о видеоигре! : D

```
# get the MONGO_URL string for your app
meteor mongo --url $METEOR_APP_URL

# then quickly copy all the info into the following command
mongodump -u username -p password --port 27017 --db meteor_app_url_com --host production-db-
b1.meteor.io
```

Экспортировать данные из местного экземпляра разработки Meteor?

Эта команда создаст каталог / dump и сохранит каждую коллекцию в отдельном файле blob BSON. Это лучший способ резервного копирования или передачи баз данных между системами.

```
mongodump --db meteor
```

Восстановить данные из Dumpfile

Аналогом команды `meteordump` является `meteorrestore` . Вы можете сделать частичный импорт, выбрав конкретную коллекцию для импорта. Особенно полезно после запуска команды drop.

```
# make sure your app is running
meteor

# then import your data
mongorestore --port 3001 --db meteor /path/to/dump

# a partial import after running > db.comments.drop()
mongorestore --port 3001 --db meteor /path/to/dump -c comments.bson
```

Экспорт коллекции в JSON

Запустите метеорит, откройте другое окно терминала и выполните следующую команду.

```
mongoexport --db meteor --collection foo --port 3001 --out foo.json
```

Импортируйте файл JSON в Meteor

Импорт в экземпляр Meteor по умолчанию довольно прост. Обратите внимание, что вы можете добавить параметр -jsonArray, если ваш json-файл экспортируется как массив из другой системы.

```
mongoimport --db meteor --port 3001 --collection foo --file foo.json
```

Копирование данных между промежуточными и локальными базами данных

Mongo поддерживает копирование базы данных в базу данных, что полезно, если у вас есть большие базы данных в промежуточной базе данных, которые вы хотите скопировать в локальный экземпляр разработки.

```
// run mongod so we can create a staging database
// note that this is a separate instance from the meteor mongo and minimongo instances
mongod

// import the json data into a staging database
// jsonArray is a useful command, particularly if you're migrating from SQL
mongoimport -d staging -c assets < data.json --jsonArray

// navigate to your application
cd myappdir

// run meteor and initiate it's database
meteor

// connect to the meteor mongodb
meteor mongo --port 3002

// copy collections from staging database into meteor database
db.copyDatabase('staging', 'meteor', 'localhost');
```

Компактная база данных Mongo на ящике Ubuntu

Предварительное выделение. Монго выделяет дисковое пространство в пустых контейнерах, поэтому, когда придет время написать что-то на диск, ему не нужно сначала перетасовывать биты. Он делает это с помощью алгоритма удвоения, всегда удваивая объем дискового пространства, предварительно распределенного до достижения 2 ГБ; а затем каждый файл prealloc оттуда составляет 2 ГБ. Как только данные будут предварительно распределены, он не будет освобожден, если вы специально не сообщите об этом. Таким образом, наблюдаемое использование пространства MongoDB имеет тенденцию к увеличению автоматически, но не вниз.

Некоторые исследования по предопределению Монго ...
восстановительно-MongoDB-базы данных размер файла
Mongo-prealloc-файлы взятие вверх-зал

```
// compact the database from within the Mongo shell
```

```
db.runCommand( { compact : 'mycollectionname' } )

// repair the database from the command line
mongod --config /usr/local/etc/mongod.conf --repair --repairpath /Volumes/X/mongo_repair --
nojournal

// or dump and re-import from the command line
mongodump -d databasename
echo 'db.dropDatabase()' | mongo databasename
mongorestore dump/databasename
```

Сбросить набор реплик

Удалите локальные файлы базы данных. Просто выйдите из оболочки Mongo, перейдите к / dbpath (везде, где вы его настроили), и удалите файлы в этом каталоге.

Подключить удаленно к Mongo Instance на * .meteor.com

Вы знали о флагом `--url` ? Очень кстати.

```
meteor mongo --url YOURSITE.meteor.com
```

Доступ к файлам журнала Mongo на локальном метеорном экземпляре

Они не легко доступны. Если вы запустите команду «meteor bundle», вы можете создать файл tar.gz, а затем запустить свое приложение вручную. При этом вы должны иметь доступ к журналам mongo ... возможно, в каталоге .meteor / db. Если вам действительно нужно получить доступ к файлам журнала mongodb, настройте обычный экземпляр mongodb, а затем подключите Meteor к внешнему экземпляру mongo, установив переменную среды MONGO_URL:

```
MONGO_URL='mongodb://user:password@host:port/databasename'
```

Как только это будет сделано, вы сможете иметь доступ к журналам в обычных местах ...

```
/var/log/mongodb/server1.log
```

Поворот файлов журнала в ящике Ubuntu

Поверните эти файлы журналов, или они в конечном итоге съедят все ваше дисковое пространство. Начните с некоторых исследований ...
MongoDB лог-файлов роста
вращать лог-файлы

Файлы журнала можно просмотреть с помощью следующей команды ...

```
ls /var/log/mongodb/
```

Но для настройки поворота лог-файла вам нужно будет сделать следующее ...

```
// put the following in the /etc/logrotate.d/mongod file
/var/log/mongo/*.log {
    daily
    rotate 30
    compress
    dateext
    missingok
    notifempty
    sharedscripts
    copytruncate
    postrotate
        /bin/kill -SIGUSR1 `cat /var/lib/mongo/mongod.lock 2> /dev/null` 2> /dev/null || true
    endscript
}

// to manually initiate a log file rotation, run from the Mongo shell
use admin
db.runCommand( { logRotate : 1 } )
```

Прочитайте Управление базой данных Mongo онлайн:
https://riptutorial.com/ru/meteor/topic/3707/управление-базой-данных-mongo

глава 48: Фоновые задания

замечания

Пакет **cron-tick** - очень простой пакет для фоновых задач, но он не поддерживает несколько процессов, если вы запускаете приложение в нескольких процессах (или в контейнерах), используйте **percolate: synced-cron** вместо этого.

Examples

Простой cron

Используйте пакет **перколяции: synced-cron**

Определить работу:

```
 SyncedCron.add({
  name: 'Find new matches for a saved user filter and send alerts',
  schedule: function(parser) {
    // parser is a later.parse object
    return parser.text('every 10 minutes');
  },
  job: function() {
    user.alerts.map(a => a.findMatchesAndAlert());
  }
});
```

Запуск определенных заданий:

```
SyncedCron.start();
```

Он поддерживает синхронизацию заданий между несколькими процессами, такими как Galaxy с более чем 1 контейнером.

Прочитайте Фоновые задания онлайн: https://riptutorial.com/ru/meteor/topic/4772/фоновые-задания

кредиты

S. No	Главы	Contributors
1	Начало работы с метеоритом	Ankit, Christian Fritz, Community, Gal Dreiman, ghybs, grahan, hwillson, João Rodrigues, levon, Matthias Eckhart, mav, mertyildiran, Ray, reoh, robfallows, Tom Coleman, Zoltan Olah
2	Blaze Templating	Dan Cramer, distalx, ghybs, jordanwillis, khem poudel, RamenChef, robfallows, Thomas Gerot
3	Electrify - компиляция Meteor как локально устанавливаемого приложения	AbigailW, JuanGesino, Nick Bull, RamenChef
4	ESLint	saimeunt
5	Meteor + React + ReactRouter	rafahoro
6	MongoDB	distalx, Dranithix, hwillson, Matthias Eckhart, robfallows, Thomas Gerot
7	Nightwatch - Конфигурация и настройка	AbigailW
8	Агрегация MongoDB	AbigailW, levon
9	Аккаунты пользователей Meteor	Barry Michael Doyle, KrisVos130
10	активы	Matthias Eckhart
11	Горизонтальное масштабирование	AbigailW
12	Доступ к машинам сборки Meteor из Windows	Tom Coleman
13	Загрузка файлов	AbigailW

14	Извлечение данных из Meteor.call	Ramil Muratov, Rolljee, Sacha
15	Издательский выпуск	AbigailW
16	Инструменты разработки	AbigailW, Ankit, Ankit Balyan, Fermuch, Ilya Lyamkin
17	Интеграция сторонних API	AbigailW
18	Использование Meteor с прокси-сервером	AbigailW, Serkan Durusoy, Tom Coleman
19	Использование полимера с метеор	Thaum Rystra
20	Использование частных метеорных пакетов на кодах	schmidsi
21	логирование	AbigailW
22	маршрутизация	Ankit, ghybs, Luna, Michael Balmes
23	Метеор + Реакция	AbigailW, aedm, corvid, ghybs, RamenChef, Teagan Atwater, zliw
24	Миграция Монго Схемы	AbigailW
25	Мобильные приложения	AbigailW, Anis D, Antti Haapala, ghybs
26	Модули ES2015 (импорт и экспорт)	reoh
27	Монгольские коллекции	AbigailW
28	Наборы реплик и осколки	AbigailW, Anis D
29	Настройка базового кода для	schmidsi

	автоматического тестирования	
30	Настройка производительности	AbigailW, RamenChef, reoh
31	Непрерывная интеграция и облака устройств (с помощью Nightwatch)	4444, AbigailW
32	Непрерывное развертывание в галактике из кодов	schmidsi
33	Обертка асинхронных методов в волокно для синхронного выполнения.	Dranithix
34	Обнаружение окружающей среды	AbigailW, ghybs
35	отладка	AbigailW, distalx
36	Офлайн-приложения	AbigailW
37	Переменные среды	AbigailW, hcvst
38	Полная установка - Mac OSX	AbigailW, RamenChef
39	Приемочное тестирование (с помощью Nightwatch)	AbigailW
40	Публикация данных	Abdelrahman Elkady, AbigailW, Chris Pena, corvid, Dair, dangsonbk, Eliezer Steinbock, Faysal Ahmed, ghybs, j6m8, Maciek, RamenChef, Ramil Muratov, robfallows, Serkan Durusoy
41	Развертывание с помощью Upstart	AbigailW, ghybs

42	Реактивный (Vars & Dictionaries)	Ankit
43	Рецепты пользовательского интерфейса Blaze (Bootstrap, No jQuery)	AbigailW, Anis D
44	Руководство для начинающих по установке Meteor 1.4 на AWS EC2	AGdev
45	Структура каталога	AbigailW, anomepani, ghybs, Michael Balmes, Nick Carson, Phe0nix, reoh, Thomas Gerot
46	Узел / НМП	hwillson
47	Управление базой данных Mongo	AbigailW, distalx, RamenChef, TechplexEngineer
48	Фоновые задания	Filipe Névola